A Database Perspective on Large Scale High-Dimensional Indexing

Laurent Amsaleg

HABILITATION À DIRIGER DES RECHERCHES

présentée devant

L'université de Rennes 1
Spécialité : informatique

par

Laurent Amsaleg

A Database Perspective on Large Scale High-Dimensional Indexing

soutenue le 20/11/2014 devant le jury composé de :

M.	Jean-Marc Jézéquel	Président
M.	Michel Crucianu	Rapporteur
M.	Tamer Özsu	Rapporteur
M.	Shin'ichi Satoh	Rapporteur
M.	Edward Chang	Examinateur
M.	Éric Diehl	Examinateur
M.	Patrick Valduriez	Examinateur
M.	Marcel Worring	Examinateur

À Emmanuelle, Vincent, Clémentine et Lise.

R., avec une moue.
Vous m'offrez du brouet quand j'espérais des crèmes !

Remerciements

Je tiens d'abord à remercier les membres du jury, en particulier Michel Crucianu, professeur au Conservatoire national des arts et métiers, Tamer Özsu, professeur à l'université de Waterloo, Canada, et Shin'ichi Satoh, professeur au *National Institute of Informatics*, Japon, qui ont accepté la lourde tâche de rapporteurs.

Je remercie ensuite Jean-Marc Jézéquel, professeur à l'université de Rennes 1 pour avoir accepté de présider ce jury.

Pour leur participation, je remercie également Edward Chang, professeur adjoint à la *Hong Kong University of Science & Technology*, Hong Kong, Éric Diehl, *VP Media and Content Security* chez *Sony Pictures Entertainment*, Patrick Valduriez, directeur de recherche INRIA et Marcel Worring, directeur associé du *Data Science Research Center* au sein de l'université d'Amsterdam.

Je ne pouvais rêver meilleur jury.

Je tiens ensuite à témoigner ma profonde gratitude à Patrick Gros, directeur de recherche INRIA pour tout ce qu'il a pu m'apporter au cours de ces nombreuses années, scientifiquement et aussi humainement. Notre aventure commune a commencé il y a bien longtemps, sur un petit coin de bureau. Quel chemin parcouru !

De très sincères remerciements vont aussi à mon cher ami Björn Þór Jónsson, avec qui j'ai tellement de plaisir à travailler, tout est si facile et si rapide. C'est pour toi, mon vieux complice, que cette habilitation est écrite en anglais. Ég sendi þér og þínum hjartans kveðjur, svo og eylandi þínu dularfulla.

Je remercie aussi tous les membres de l'équipe TexMex, stagiaires, doctorants, ingénieurs, assistants, enseignants et chercheurs, post-doctorants, invités, avec qui j'ai eu le plaisir de travailler toutes ces années. Certains n'ont fait que trop brièvement traverser l'équipe. L'idée de continuer ensemble au sein de LinkMédia me fait très plaisir et est une perspective extrêmement motivante. Merci à Guillaume Gravier d'en avoir pris la tête.

Je tiens à remercier Christine Guillemot, Dominique Lavenier et Gilles Lesventes. J'adresse aussi un salut particulier à Michael E. Houle dont j'apprécie l'esprit toujours en ébullition dans cet étonnant Japon.

Je remercie aussi tous les gens liés au support des équipes de recherche au sein de l'IRISA et d'INRIA RBA, qui œuvrent pour faciliter notre quotidien. Je remercie en particulier toutes les personnes faisant un boulot si formidable aux relations internationales d'INRIA, du CNRS ou encore d'Égide.

Il y a aussi Aglaé, Alexis, Aloïs, André, Annaëlle, Bastien, Catherine, Christine, Christophe, Célia, Élie, Éric, Ewen, Fabienne, Félix, Inès, Karine, Lucie, Manon, Michel, Nathalie, Nicolas, Nolan, Noémie, Sandra, Stéphane, Thomas.

Enfin, je remercie Emmanuelle, Vincent, Clémentine et Lise pour leur gentillesse, leur affection, leurs passions et tout ce qui fait que la vie avec vous est belle.

Contents

List of Figures

List of Tables

Chapter I

Introduction

Contents

It is a truism to say we observe an extraordinary growth in the volume of digital multimedia material that is created and made available every day. People take pictures, shoot videos, record audio, and all this can easily be shared on many popular platforms collecting user generated contents. Multimedia material is also massively produced by professional players such as TV and radio broadcast channels or movie companies. This material is not only produced, it is also captured and recorded by other parties, such as archivists, who in turn produce and collect new material.

This material takes all its cultural and economic value, all its artistic wonder when it can be accessed, watched, searched, browsed, visualized, summarized, classified, shared, ... This allows users like you and me to fully enjoy the incalculable richness of the collections. It also makes it possible for companies to create business rooted in this multimedia material with the goal of increasing users digital experience enjoyment.

Without any loss of generality, *search engines* can help finding the multimedia contents one user wants to access (this is the needle) in the vast amount of existing documents that can be

accessed (this is the haystack). Today, the vast majority of search engines eventually returning multimedia material ask their users to provide a query consisting in a few words, and then return from their databases means for accessing the documents that are the most relevant to these querying words. This search paradigm centrally relies on text, and the text attached to multimedia documents typically refers to titles, names of actors/authors, possibly to a short, manually crafted, free-text description.

Text-based search engines work extremely well. They work at very large scale, they are incredibly fast and good and their inventors, in-the-large, must be thanked. There is, however, a downside when so much relying on text. Entering text and annotating multimedia collections remain a manual burden, especially when these textual annotations become semantically rich, which is desirable for having search engines returning highly relevant material. But maybe a more severe downside is that text forces users to query with words, which is sometimes not the most appropriate search paradigm for multimedia.

For that reason, *querying by contents* has emerged as an alternative search paradigm where an example of what the user is interested into is given to the search engine. For multimedia, this usually takes the form of an image, or of a short video/audio sequence. Then, the search engine tries to identify in its collections the multimedia material that is the *most similar* to the one given.

Content-based retrieval systems implementing this type of search engines include specific algorithms for automatically describing the multimedia material as well as for running searches. The descriptions typically take the form of high-dimensional features vectors extracted from the multimedia items. The descriptions somehow capture the visual contents of the images, of the videos, they capture the aural contents of audio tracks, etc. Descriptions come with a measure allowing to determine how similar two features are. Comparing features vectors and assessing their similarity is the very fundamental operation at the core of content-based retrieval systems. It is assumed that the similarity between multimedia documents is directly related to the similarity of their feature vectors. Therefore, two documents are said to be similar if their descriptions are close in the high-dimensional space. The literature contains many contributions for best capturing into features the correct visual/audio/... information from contents, for a particular application domain.

Once it is possible to compare two features, it is natural to compare the features extracted from one multimedia document to the ones extracted from an entire collection of documents stored in a database in order to discover the most similar ones; it is natural to try to cluster features, grouping together features that are highly similar while separating in different groups features that are less similar, which, in turn, allows to cluster documents, to classify them, etc.

When it turns to considering the comparison of features, the most fundamental core operation run inside every single content-based retrieval system is probably the one that implements *similarity searching*. It is typically either an ε-search or a k-nearest neighbor search. An ε-search identifies all the feature vectors that are within an ε distance range from a particular query feature vector. A k-nearest neighbor search, or k-nn search, identifies the k feature vectors that are the closest to a particular query feature vector.

To be usable in practice, content-based retrieval systems must be fast when it comes to run similarity searches. High-dimensional indexing techniques therefore play a central role here, as their design and implementation allow very efficient searches over the database storing multimedia material.

More than a decade of very active and prolific research lead to the publication of many elegant contributions overall spanning a quite large spectrum of multimedia applications dealing with still images, videos, audio, sometimes considering multimodality. Furthermore, the maturity of

these techniques now allows startups and major software companies to build multimedia search engines that are profitable, money wise. Good performance are here today: systems are fast, and the quality of the results they return is excellent. While near the turn of the millennium systems were managing few thousands images, few dozen hours of videos, we can now find systems that cope with millions images, thousands hours of video; accuracy is extremely good; fine grain recognition of objects, of person, in images are possible; response times are such that systems can be used in practice.

We are however quite far from having content-based retrieval systems that can cope with several billion still images, that can search within collections of archived videos comprising millions of hours, that can classify web-scale collections. Furthermore, aside this searching problem, we are far from being able to store all the feature vectors from this material in a failure-resistant safe manner and manage this data kept on secondary storage with decent performance. We are far from being able to cope with the dynamicity of collections that grow at a fast pace, making sure content-based search engines probe up-to-date collections. Flickr, a popular image sharing site, probably stores more than 6 billion high-resolution images, and grows by roughly 1.5 million every day; Facebook claims to store about 1,000 billion pictures and their collections grows by 1.4 billion every single week.

No system is currently able to cope with this, no system can today identify duplicated images in Facebook's collections in a decent amount of time. No system can process the million hours of videos archived by the TV/radio French archive institute (INA) to facilitate their browsing, to allow detecting similar elements, to find the documents where a particular individual is, where a specific topic is evoked. No system can filter in real time the deluge of videos uploaded every minute on YouTube in order to track illegal material such as terrorist propaganda or to enforce the protection of the copyright of Works.

The quest for improving speed, quality, scale and performance remains there.

This manuscript contains an overview of more than 10 years of research aiming at inventing and evaluating systems and techniques for building efficient content-based retrieval systems managing extremely large databases of multimedia material. Many top research teams worldwide have also worked on this topic, and we all together participated in improving the comprehension of the field. This manuscript describes several routes we took to discover elements of understanding, to then design and implement techniques described in publications but either kept inside our research lab or transferred to partners in charge of building systems usable in the real world.

Overview of the Chapter

To travel along that route, and before entering the technical chapters, this first chapter will set the plot. It is crucial to clarify that we have mainly focused our work on managing large collections of *images*; nevertheless, quite a lot of the ideas, of the concepts, of the discoveries discussed here also apply to multimedia in general.

We will thus overview in this chapter the main components of content-based image retrieval systems and briefly introduce the role of indexing techniques. Then, we will very quickly explain why traditional high-dimensional indexing techniques become inefficient when they have to manage extremely large scale collections of multimedia material. This discussion will be followed by the description, at very high-level, of the main approaches researchers invented to address scale issues. We will then define and motivate what is a database perspective on large scale high-dimensional indexing. Next, we will be giving a rough list of the main contributions detailed in this work and we will go through the structure of this manuscript. This chapter ends with a discussion paying homage to co-workers.

Note that, however, this manuscript is not proposing to the reader a complete state-of-the-art section reviewing the seminal high-dimensional indexing problems and solutions. This has been done in parts in our publications and this will not be repeated here. Of course, elements of background work will be given wherever this will be necessary for the correct understanding of our work.

I.1. Content-Based Image Retrieval Systems

Overall, the workflow when using a content-based image retrieval (CBIR) system is as follows: a CBIR system must first process off-line a large collection of images to extract an also quite large number of features from the images. Then, these features are stored in a database, indexed for efficiency, after which the system is ready to get on-line queries from users. A user queries the system by submitting an image. This query is processed to extract its features, and these features are used to identify the most similar images from the database, eventually returned to the user.

A CBIR system implementing this workflow is thus basically made of two large blocks of software. The first block is in charge of extracting the high-dimensional features from the images. The second block is for indexing and searching: it is in charge of storing these features inside a particular data structure designed to efficiently run subsequent similarity searches. The work in this manuscript focuses on this second block. Means for describing images deserve however a short description, provided below.

Capturing Contents for Image Similarity

Many approaches have been proposed to capture the contents of images in order to assess their similarity [Szeliski, 2010, Datta et al., 2008, Sebe et al., 2008]. They all boil down to transforming some of the visual elements that exist in images into high-dimensional vectors, used together with a distance function allowing to evaluate their similarity. The description power of high-dimensional descriptors is often linked to the dimensionality of the vectors. The trend is to design very high-dimensional descriptors for an improved description power. Dimensionality of choice typically goes over the hundred and can reach several thousands or hundred thousands in some specific cases. Comparing descriptors is typically done via an Euclidean distance function.[1]

Current image description schemes are termed *global* or *local* depending on the number of features they compute per image. Global description schemes compute one feature per image that somehow captures the visual information from the entire image. In contrast, local description schemes compute a varying number of features per image and each feature captures the visual information that is local to where the feature is computed. SIFT is a good example of a local description scheme, see [Lowe, 2004]; it is used almost everywhere and became a de facto standard. Please refer to [Szeliski, 2010] to get a nice survey of local description schemes. What matters the most here is the way the distance functions defined for global or local schemes are typically used. It is very often the k-nn search approach that is used, even though the other ε-search technique proves to be useful in many cases. We focus on k-nn in this manuscript.

[1] This paragraph purposely presents an over-simplified view of features and means to compare them. Reality is much more complex and computer vision processes for image similarity are way more sophisticated. Some features are defined in vector spaces, compared with distance functions, other features are not vectors and exist only in metric spaces, some other approaches use similarity functions that are not distances; norms different than 2 can be used; dynamic programming distances also, such as the EMD. We focus on L_2 and features defined in a vector space.

When a CBIR system manages a database of images described using global features, then one global feature is computed from the query image, and this feature is used to identify the k closest features from the database. Since each feature identifies the image it comes from, it is then straightforward to return to the user the k most similar images. In this case, the distance function coming together with the descriptors is directly used to identify the most similar images.

In contrast, that search process is slightly more complicated when it comes to using local features. In this case, many features are extracted from every single image of the collection to search, and all these features are kept in the database. At query time, many features are extracted from the query image and a k-nn search process is ran for each of these features. All the (close) vectors hence found in the database then vote for the images they have been generated from. This allows to rank the images from the database according to that voting score, and the images with the highest scores are returned. It should be extremely clear that in this case the distance function on features is used to determine a set of close vectors from the database, and that a *second* similarity function is used to subsequently identify the most similar images—typically ranking based on votes.

Indexing for Efficient Similarity Retrievals

The second software block of a CBIR system is dedicated to indexing for efficiently running similarity retrievals. The following text about retrievals that is in this introduction is short; the entire manuscript is dedicated to this topic.

When the collections are large, then it is mandatory to insert the features describing the images in a particular data structure allowing very fast searches. This is indexing. Designing an high-dimensional indexing technique means inventing the data structure as well as the search process that efficiently goes through this structure.

We sketched above the basic workflow for a CBIR system. This translates into off-line creating an index with all the features computed on the images from the collection, and then on-line searching that index at query time for similar elements. There are therefore two parts in high-dimensional indexing techniques, one being dedicated to the creation of the index, the other being dedicated to searching the index. Of course, they are designed in a combined manner.

When discussing the contributions we made in the field of high-dimensional indexing, this manuscript will typically present the index creation process and then the index searching process.

Assessing the Performance of CBIR Systems

In order to assess the performance of CBIR systems, several different points of view can be taken. The first one is related to the quality of the results a particular system is returning, how good it is at finding the elements that are similar to a query. Another very important criteria is its response time, as an impatient user is often waiting for the answers.

The performance of a CBIR system is typically governed by the size of the collection it indexes. When it is too large to fit in RAM and/or when being resilient to failures is necessary, then measuring the costs for performing I/Os is key to performance. I/Os may become a severe bottleneck. Ignoring disks, a large collection tends to require performing a lot of distance calculations in order to identify the k-nn of a query point. The cost of these calculations is particularly prevalent when the dimensionality of the descriptors is high, since many dimensions are involved. CPU may become a severe bottleneck.

We dedicate an entire chapter to the evaluation of CBIR systems.

I.2. High-Dimensional Indexing

Over the years, many high-dimensional indexing techniques have been proposed in the literature. Early works originates from the seminal R-Tree [Guttman, 1984] and K-D-Tree [Bentley, 1975]. They respectively proposed data partitioning and space partitioning techniques splitting the high-dimensional data collection into cells. The search process aimed at analyzing only a portion of all existing cells which, in turn, provided efficiency. A very long series of contributions enhancing these two approaches exists. The interested reader can refer to excellent surveys (see for example [Böhm et al., 2001] and [Datta et al., 2008]) as well as a quite complete book by Samet [Samet, 2007].

One key property all the early indexing schemes have, is that they guarantee to identify the exact k-nn of a point in the high-dimensional space. To provide that guarantee, the search process has to scan every cell that may contain a neighbor possibly closer to the query point than are all the neighbors already found. Enforcing this guarantee turned out to severely limit the performance of all the proposed high-dimensional schemes. Extensive work was carried out to understand why the performance were limited, and it soon became clear that existing indexing schemes were suffering from the dimensionality curse problem. These problems are nicely surveyed in [Böhm et al., 2001] and in [Berrani, 2004, Berrani et al., 2002].

With their study, Beyer et al. demonstrated in 1999 (see [Beyer et al., 1999]) that no *exact* high-dimensional indexing technique can remain efficient at scale, that is, when the dimensionality of the vectors that are indexed becomes sufficiently high and/or when the number of vectors to index becomes large enough. Alternative indexing solutions had to be invented to address scale issues.

I.3. Addressing Scale

To cope with scale issues, i.e., to build CBIR systems that can manage extremely large collections of highly dimensional vectors, researchers basically investigated the following directions detailed thereafter:

1. *Changing the Definition of the Problem: from Exact to Approximate Indexing.*
2. *Relying on the Definition of the Application: Aggregating Local Descriptors for Image-Level Recognition.*
3. *Relying on Hardware and Modern Programming Frameworks.*

I.3.1. Changing the Definition of the Problem: from Exact to Approximate Indexing

Guaranteeing the correctness of the k-nn search process together with the dimensionality curse problems cause exact high-dimensional indexing techniques to become inefficient at scale. When the dimensionality of the indexed vectors is high, then enforcing this correctness property compels the search process to analyze almost all the cells, hence, almost all the vectors stored in the database, which becomes very costly when collections are large.

Indeed, research studies have demonstrated that most of the nearest neighbors an exact k-nn search will eventually return are found very early during the search process, after having analyzed only a fraction of the database. This motivated researchers to investigate what is

at stake when the correctness guarantee is relaxed, trading-off speed for a reduced quality of the results. Then came the time where *approximate* high-dimensional indexing strategies were proposed. Instead of returning the true k-nn of a query point, the resulting approaches return k close neighbors, and not the closest ones.

The game then played by researchers was to invent techniques returning a set of close neighbors that is as much as good as the set of closest neighbors. Some neighbors might be missed, some not-that-close neighbors might be returned. Overall, there is for sure a loss in the quality of the results, that researchers try to minimize, but this is totally compensated by the dramatic improvement of performance.

Having changed the definition of the problem allowed high-dimensional techniques to make gigantic progresses [Indyk and Motwani, 1998, Lejsek et al., 2011, Sivic and Zisserman, 2003, Jégou et al., 2011]. It became possible to at last go beyond the ridiculously small scale at which existing systems could only work, to have amazingly fast search systems allowing companies to seriously plan business, etc.

All the work in this manuscript follows that route. All the techniques we are going to detail in the next pages are based on approximate high-dimensional indexing schemes. Because they are approximate, then it is absolutely mandatory to quantify how much result quality decreases compared to an exact solution. For that reason, there is a strong emphasis in this manuscript on benchmarking and on evaluations, and traditional metrics like the precision and the recall indicators computed on search results are systematically used.

I.3.2. Relying on the Definition of the Application: Aggregating Local Descriptors for Image-Level Recognition

Scale problems became particularly prevalent when (exact or approximate) high-dimensional indexing techniques had to deal with image collections described using local description schemes such as SIFT [Lowe, 2004].

A first problem comes from the very large number of descriptors that are typically created when describing an image collection with local descriptors because as much as several thousand descriptors can be generated from one strongly textured image. This overall dramatically increases the size of the database: instead of storing one descriptor per image, as it is the case with global descriptors, many local descriptors must be stored for each image. The size of the database is typically increased by two order of magnitude. As a result, searches typically take longer to run.

That large number of descriptors also impacts the search process: instead of searching the descriptors that are close to the unique query descriptor (as it is the case with global descriptors), the system has to query the database many times, each time using a different local query descriptor. Typically, several hundred to a few thousand consecutive queries probe the (very large) database when querying with a standard image. This demanding process increases the impact of the performance problems traditional high-dimensional index techniques suffer from.

The highly variable number of descriptors in each query impacts the duration of the search process: it is faster when there are few query descriptors. A very large variance in the response times of the system can therefore be observed when using local descriptors. This may negatively impact some applications.

The great quantity of local descriptors, on the other hand, allows for extremely accurate image recognition. It is possible to identify images that are similar from few *details*. The recognition power of local descriptors is so good that they are used in almost every single application con-

cerned with image similarity. With local descriptors, approximate high-dimensional strategies work very well. On the one hand approximations make things to run fast as true nearest neighbors might be missed; on the other hand, missing few true neighbors among thousands does not matter, it is almost transparent in terms of quality.

There is one class of applications that is typically interested into detecting quasi-copies of images. This is for example useful for finding duplicates in an image collection. In this case, it is the overall similarity of pictures that is of interest at the level of the application. It is very often local description schemes that are used since their recognition power is extremely good, even if applications are not concerned with identifying similar images from tiny details. A global descriptor would not do the work, it is not as good, recognition wise. Global descriptors, however, have the desirable properties to create a much smaller database compared to the local description schemes. There is only one global descriptor per image instead of several hundreds, and it requires only one k-nn search per query image. But in turn if the good neighbors are missed due to the approximate searching scheme, then recognition suffers; there is no redundancy to compensate misses, as it is the case with local schemes.

To retain at the same time most of the good recognition power of the local descriptions and to reduce the size of the database and the complexity of running queries, researchers invented ways to aggregate the many local descriptors of one image into a unique signature for that image. Such aggregated descriptions are typically of a much higher dimensionality compared to the original descriptors they are computed from. Indexing aggregated descriptions instead of local ones creates a database of a much smaller cardinality, the search process probes the database only once per query image, fixing the variance of response times. In turn, some recognition accuracy is lost, and it becomes hard if not impossible to identify images from tiny details. But this does not matter for such applications, they are concerned with quasi-copies and with the identification of similar visual elements at the level of images.

With such approaches, processing one million images described with SIFT requires to index "only" one million aggregated descriptors, instead of about one billion if they were kept non aggregated at indexing time. For example, Jégou et al. [Jégou et al., 2010a] can index 10 million images in main memory, partly because of local descriptors aggregation. The elegant aggregation schemes used are such that the resulting data collections fit in main memory and do not require accessing disks. In turn, their performance are impressive, they can return answers in milliseconds. This is hardly achievable when the descriptors collections are kept on disks due to scarce RAM because costly I/Os always severely impact response time.

The work in this manuscript considers the two cases when descriptors are aggregated and when they are not aggregated. We started investigating the design of high-dimensional indexing techniques before the wide spreading of aggregation techniques. Therefore, a large portion of our contributions do not aggregate descriptors, making it possible to use our techniques to build CBIR systems able to search similar details in images as well as for doing image-level recognition.

I.3.3. Relying on Hardware and Modern Programming Frameworks

In order to address scale issues, researchers have designed many sophisticated algorithms having a complexity that is reduced again and again, they have slightly changed the definition of the problems, as we discussed above. There is also another family of contributions where the power of the computers that are used to implement CBIR systems is the central focus of attention.

The majority of the state-of-the-art high-dimensional indexing solutions are implemented over

a centralized single-core architecture. In fact, hardware concerns are often not discussed at all, assuming a one machine setting. In contrast, there are few contributions that specifically take into account specific hardware architectures. This, in turn, makes it possible to run systems at very large scales, even if the fundamental indexing algorithms they run are not that sophisticated. It is the joint power of multiple processing units that give performance.

All machines are now with 64 bits addressing capabilities, hence the amount of RAM that can be installed in a computer is virtually not limited (but by the cost of memory chips). Machines by default are all multi-cores, clusters of computers are not that uncommon and available in many places, their aggregated memory being possibly very large. In the past, writing parallel and/or distributed programs was known to provide efficiency, but also major complications for synchronizing the independent threads/tasks, resisting failures, balancing the load, etc. The development of very powerful and convenient operating system tools and frameworks during the last decade completely changed the picture. While parallel and distributed programming was reserved to some elite, equipped with exceptional machines, it is now relatively easy thanks to environment such as the Hadoop framework and the Map-Reduce programming model.[2]

It is not uncommon anymore to find in the high-dimensional indexing literature contributions using such frameworks. Relying on the power of computers is really beneficial when addressing scale issues. Despite the frameworks, it is not trivial to port a centralized indexing/searching solution to parallel and distributed settings.

This manuscript follows that route too. We will extensively discuss a parallel and distributed high-dimensional indexing strategy, and explain what are the obstacles along the way.

I.4. A Database Perspective

Most of the work described in this manuscript has been achieved while having in mind a database perspective. This section defines this perspective.

The most important notion that shades a particular light on the high-dimensional indexing techniques we have been developing is the *mandatory need for secondary storage*. There are several reasons for designing indexing techniques where disks are a central concern.

Disks for Absorbing Failures. First of all, pushing data to stable storage is today the only mechanism we know to best cope with failures. Absorbing failures and recovering from crashes is one of the hardest component to design and implement in a database management system. It must gracefully handle system crashes where RAM is lost, as well as cope with more catastrophic media failures where stable storage gets corrupted.

When considering CBIR systems, a failure means loosing the indexed data. If only the contents of RAM is lost, then a CBIR system can restart once data is fetched again from disk. This obviously means that this contents had to be pushed to disks at one point in time. Doing it properly is extremely complicated. If the indexed data was not on disks, then the un-indexed high-dimensional descriptors must be read again from disks before being re-indexed again. When this is a very time consuming process due to the scale of the collection, doing it again may be seriously problematic. Note that re-indexing is totally unacceptable for production systems running in the real life, they can not disrupt their service, this is too costly.

[2]Note that there are other nice frameworks facilitating the implementation of parallel/distributed programs, such as GraphLab [Low et al., 2012], Dryad [Isard et al., 2007], Shark/Spark [Engle et al., 2012, Zaharia et al., 2010]. Hadoop is probably the most popular, however.

Worse, it might be that the lost high-dimensional vectors have to be re-extracted from the image collection. They might have been lost during the crash or might never have been stored on disks... Re-extracting features is fine for most prototypes developed in research labs; it is less acceptable when such an extraction consumes days, weeks or months of computing cycles as it is the case when indexing several hundreds of millions of images; and it is not an option for systems running in the real life. Obviously, features can be re-created only if the images themselves have been safely stored somewhere. Note that this might not be possible when features have been outsourced to a third party. The rationale for outsourcing is that this third party has enough resources to do the indexing and to run the searches, in contrast to the owner of images that has too scarce resources. In the event of a severe crash, then this third party has no way to re-create the features but to ask the owner again; alternatively, it has to take special care to preserve the high-dimensional vectors to survive crashes.

These very realistic and pragmatic problems motivate the compulsory need to add disks to cope with failures, which in turn forces us to understand how disks impact CBIR systems. We aim at designing high-dimensional indexing solutions that may be transferred to industry, hence, preserving data across failures is mandatory. Pushing data to disks is extremely complicated. It requires way more sophistication than simply opening a file in write mode and dumping memory to it. It has to survive media crashes, allow fast recovery, handle failures *while* recovering, ... Over the years, database researchers worked crash recovery so much that it became conceptually simple, Aries [Mohan et al., 1992] being the best example. For all these reasons, we can not ignore durability issues. This is central to our database perspective.

Disks for Dynamicity. Pushing data to disks is also crucial when the collection that is indexed is dynamic, when new images are inserted while the system is running. Most indexing techniques in research labs simply ignore this issue, or are fine with halting the system and then re-creating an entirely new index over the augmented image collection. This is fine for academic prototypes, but this is not acceptable for systems that are closer to real life requirements. Having a database perspective means here that dynamicity must be addressed, that the updated collection must be made durable across failures and that updates have to be enforced while the system is up and running. Note that this may cause possible consistency issues in the data or in the index, or in both.

Disks for Handling Scale. Using disks is also motivated by the fact that, eventually, there will be too much data for keeping it in RAM. While there are today some very sophisticated indexing techniques that compress information such that current very large collections of high-dimensional features can stay in main memory, this can not be pushed orders of magnitude further. Eventually, data has to be pushed to disks because no system has enough RAM. It is true that the aggregated memory of distributed systems can get virtually as large as it is needed to fit in RAM an extremely large collection. In this case, however, the increased amount of hardware components increases the likelihood of failures, which, in turn, emphasizes the need for disks. We want to cope with billions of high-dimensional descriptions, with multiple tera-bytes of storage for the features, and, at that scale, disks are needed.

Response Time vs. Throughput. The default relationships a user has with a CBIR system are on-line interactions, where the user submits a query, quickly gets an answer and possibly repeats this interaction loop. This is fine for a very large set of applications. The history of databases shows optimizing for response time is not the only possible option, and that optimizing for

throughput is also key for a large family of applications. Optimizing for throughput is batching. In this case, many queries are processed at once. This might be beneficial for applications where response time is less important than throughput. Batching is also relevant in situations where no users explicitly exists, such as in the cases where a CBIR system is used to filter out duplicated images in a collection, for example. Therefore, our work will consider both optimization goals, and sometimes our contributions will focus on response time, sometimes on throughput.

Facilitating Resource Consumption Planning. Database management systems are in part successful because they do an excellent job at optimizing SQL queries. Optimization uses cost models that overall facilitate the planning of resource consumption, which in turn is best for performance. This is particularly true when the actual cost for running a task matches quite closely the cost estimated at optimization time. When there is a large difference, then the plan for making best use of the computing resources turns out to be far from optimal.

There is no such optimization in the context of CBIR systems, there is no algebra, no commutativity and/or associativity rules to best play with the order of operations. Nevertheless, it is very important for people using CBIR systems in the real world to know how much resources are needed to run similarity searches in order to correctly set the size of the system. What is the best size for RAM, what is the best number of CPU cores, what is going to be the typical response time for a search, the typical throughput of the system... In contrast to database management systems where this can be somehow computed from cost models, beforehand, it is instead more likely to be empirically observed in the case of CBIR systems.

A database perspective on resource consumption suggests to come up with a system that is as much balanced as possible. There should be no dramatic differences in the consumption of computing resources between any two queries that do not conceptually differ, e.g., two k-nn searches where k is fixed should consume roughly the same amount of resources, regardless of their respective query points. If we empirically observe that two such queries have very different costs, than any resource consumption planning is hard. This, in turn, complicates the design of a CBIR that need to reach specific goals such as meeting throughput requirements for processing the requests from users.

Finally, note that batching allows to optimize the processing of many queries since analyzing all of them at once facilitates resource planning. Typically, this allows to define an order for running the searches that is the most profitable. Batching also helps optimizing the construction of the index, and bulk loading procedures are often used.

Generic Enough to Fit Diverse Applications. Databases and the relational model proved to be able to fit a very large variety of applications, with quite different specific characteristics. A database management system can somehow adapt to this variety, and no deep changes in its internal design are required. We also want to pursue this genericity property with our high-dimensional indexing contributions. We do not want our techniques to be totally tied to the specific properties of a single application, where everything is extremely well optimized but completely frozen in the mold. For example, there are some indexing techniques that hard-wire recognition at the level of entire images; in contrast, we want our techniques able to run the most fundamental operation which is to return the k-nn of query points, ignoring what this similarity is used for in subsequent steps.

Watching for Bottlenecks. Watching for bottlenecks is a concern for all computer programmers, but it has a particular flavor in the domain of databases. Applications built over databases

are so intensive, shuffle so much data, that taking great care of the two major bottlenecks is a central concern. It is obvious that a lot of attention must be dedicated to avoiding as much as possible the disk I/O bottleneck. But at the same time, making sure the system does not become CPU bounded is also key. Note that, however, in contrast to traditional databases, no "query optimization" process applies here since there is nothing like a first order logic algebra to play with. Nevertheless, watching for both bottleneck somehow simultaneously is very important. Our desire is to care for bottlenecks during the early stages of the design of high-dimensional indexing techniques, and not as an after-thought. We do not want to design a nice technique and then add to it patches for better dealing with disks. This neither works very well nor is elegant.

Impact of the DB Perspective on the Work Presented in this Manuscript

Overall, having that database perspective on our work deeply biased our contributions. When *purposely* dealing with secondary storage, it is hard if not impossible to beat high-dimensional indexing solutions that have been designed to run in main memory. This is unfortunately true even for collections that might fit in main memory, as enforcing durability etc. generates a substantial overhead.

It is very important to realize that we are trying to understand and solve the problem of high-dimensional indexing with data on disks. It would not be fair to compare our work with other non-disk oriented solutions, both in terms of recognition quality and of speed for the system. Because of this disk orientation, there are things we can not do. For example, some approximate high-dimensional indexing techniques very significantly improve the quality of their results by analyzing large amounts of data in RAM; this is impossible as soon as disks are involved, the price to pay for so many I/Os would be too high.

I.5. Contributions and Structure

This manuscript describes several contributions.

1. *Definition of a Contrast-Based Ground Truth.* We have been working with approximate CBIR systems for some years, and we ran many evaluation campaigns to evaluate how good these systems were. We discovered that the traditional benchmarking simply based on an ε-range distance or on a simple k-nn criterion was not adequate when dealing with large scale systems. We therefore designed a contrast-based approach for building a ground truth against which systems should be evaluated. This is described in Chapter II.

2. *The NV-Tree.* Our major contribution is the NV-Tree. The NV-Tree is a high-dimensional index structure based on a scalar quantization process. It has been specifically designed from a database perspective: it can handle extremely large collections of descriptors and therefore it includes specific procedures to efficiently deal with secondary storage accesses. It also tries to mitigate the impact of the CPU bottleneck by avoiding costly distance calculations. The collections the NV-Tree indexes can also be concurrently updated while the system is running. It also deals with fault tolerance and includes specific mechanisms to enforce failure recovery.

   This contribution gave birth to a start-up company, *Videntifier Technologies*, which currently has more than 10 employees and is one of the main players in the forensics arena

where tracking down illegal digital contents is paramount. Their search engine, based on the NV-Tree, is currently able to index and identify videos from a collection of nearly 100 thousand hours of video. This work is described in Chapter III.

3. *DeCP: a Disk-Aware, Parallel and Distributed Indexing Scheme.* We designed a complete unstructured vectorial high-dimensional indexing approach harnessing as much processing power as possible from using parallel and/or distributed computing. The approach we propose truly copes with very large scale datasets, typically hundred million images, few dozen billion descriptors, tera-bytes of data. We propose in Chapter V a multi-threaded version of DeCP to harness the power of the multiple cores we find nowadays in every machine. We go one step further and then propose a Map-Reduce based version that runs on top of the Hadoop distributed framework.

4. *Elements of Understanding.* We provide discussions in Chapters II, III, IV and V that in part help understanding what is at stake when doing high-dimensional indexing from a database perspective.

Chapter VI lists some perspectives rooted in this work. It gives perspectives that are going beyond the strict limits of high-dimensional indexing as it discusses result quality improvements, the need for visualizing and organizing collections of multimedia documents, the new security and privacy problems CBIR systems have to face now as well as ways to handle multimedia sequences.

Chapter VII finally concludes this manuscript.

I.6. Discussion

This manuscript contains new and original written material, graphs, tables, figures, as well as bits and pieces extracted from some of our publications and done with co-authors. Regardless of its provenance, all the material in this manuscript is the result of interacting with many people, throwing ideas at each other, chatting in nice restaurants or in slightly more bizarre places, running experiments and analyzing results together, discussing reasons for failures or success, trying to understand why this and not that, covering boards with diagrams and drawings. This discussion aims at unambiguously paying homage to the people I had the luck to work with.

I am most grateful to Patrick Gros, Björn Þór Jónsson and Herwig Lejsek.

I am indebted to Armen Aghasaryan, Friðrik Heiðar Ásmundsson, Raghavendran Balu, Patrick Bas, Sid-Ahmed Berrani, Battista Biggio, Julien Bringer, Sébastien Campion, Oussama Chelly, Vincent Claveau, Kristleifur Daðason, Jonathan Delhumeau, Agni Delvignioti, Thanh-Toan Do, Matthijs Douze, Sigurður Holdahl Einarsson, Teddy Furon, Stéphane Girard, Philippe-Henri Gosselin, Guillaume Gravier, Ragnheiður Ýr Grétarsdóttir, David Gross-Amblard, Gylfi Þór Guðmundsson, Panagiotis E. Hadjidoukas, Kári Harðarson, Hlynur Hauksson, Michael E. Houle, Ichiro Ide, Hervé Jégou, Baldur Jóhannesson, Ársaell Þór Jóhannsson, Alexis Joly, Ewa Kijak, Dimitre Kostadinov, Ævar Kvaran, Rémi Landais, Dominique Lavenier, Julien Law-To, Bouzid Makram, Stéphane Marchand-Maillet, Arthur Masson, Benjamin Mathon, Diana Moise, Annie Morin, Michael Nett, Vincent Oria, Arnar Ólafsson, Loïc Paulevé, Patrick Pérez, Karen Pinel-Sauvagnat, François Poulet, Cordelia Schmid, Pascale Sébillot, Denis Shestakov, Michel Scholl, Hlynur Sigurþórsson, Rut Sigurðardottir, Stefán Freyr Stefánsson, Romain Tavenard, Grímur Tómasson, Hrönn Þormóðsdóttir, Li Weng, Pavel Zezula, Arthur Zimek.

Chapter II

Benchmarking: Datasets and Ground Truths

Contents

This chapter is presenting the metrics, the datasets and the ground truths we use when performing all the experiments that are described in this manuscript. We first describe the metrics we typically use. We then motivate the need for defining a contrast-based single descriptor dataset that we use to evaluate the ability of indexing methods to address the problem of performing k-nn searches in high-dimensional spaces. We then move to discussing a slightly different problem where it is image recognition using local features that matters. We therefore describe the image datasets and image ground truths we have defined as well as the quasi-copy detection scenario we use.

II.1. Metrics

In this document, several metrics are used to gauge how good indexing schemes are. Metrics can be grouped in two families, depending on whether they reflect the efficiency of indexing schemes, i.e. informally their *speed*, or effectiveness, i.e. informally their *quality*.

Usual metrics such as time (`wall clock`, `user`, and `system time`), cache misses, page faults, number of I/Os, main memory footprint, secondary storage footprint, number of queries per second, etc. are all related to the speed of the system and to the resources it consumes for answering queries from users. Depending on the application, it might matter to minimize the *response time* of searches or to maximize the *throughput* of the system. Depending on the experiments, these metrics are used here and there in this manuscript. We always clearly define the environment and the measures reported for all experiments. We very often report time-related information as well as very often observe the behavior of a particular indexing scheme from the point of view of I/Os and main memory cache misses.

Evaluating the quality of searches is particularly important, especially because almost all indexing solutions that work at scale are approximate since quality is traded-off against speed. The traditional way to assess the quality of a system is to define a ground truth from which it is easy to compute the well known precision and recall measures. From these measures, indicators such as the *mean average precision (mAP)*, the *recall@x (R@x)*, the *precision@x (P@x)*, the *ROC* curve, etc. can easily be computed. They are used every now and then in this manuscript.

It is essential to design approximate techniques with good recall; the result must contain most of what would be returned in an exact answer. Achieving good recall is often possible, at a cost of lower precision, by accepting false positive as part of the result. For some applications, false positives are acceptable; for other applications, an extra step post-processing the short list of candidates determined using a (primary) similarity measure has to be performed. This step tries to eliminate the false positives, in order to enhance the quality of the results. Traditionally, this is often enforced by using methods evaluating how geometrically consistent candidate images are with the query; this is all Hough transform and Ransac-based approaches [Fischler and Bolles, 1981, Chum and Matas, 2008]. No such mechanisms are used anywhere in the bulk of this manuscript when discussing experimental results. Note that, however, Chapter VI and its Section VI.1 that are discussing perspectives present a false positive elimination technique relying on the notion of shared and reciprocal nearest neighbors. This technique is still in its infancy and needs much more work.

II.2. Motivations for Two Types of Benchmarks: at the Level of Descriptors and at the Level of Images

Computing result quality requires the definition of a *ground truth set* against which the results are compared. Evaluating the quality of indexing schemes at the level of individual descriptors is a good indicator of their performance. It shows how capable an index is to return the good neighbors when only one descriptor is used, as it is the case when running searches for similar images with global descriptors. Such descriptors can naturally be global, as for example when they are simple color histograms. Such descriptors can also be the outcome of an aggregation process where multiple local descriptions are turned one way or the other into a unique signature. This is for example the case for "bag-of-features" approaches (see Section IV.3.1).

Overall, observing the quality of a system from a single descriptor perspective puts a very strong pressure on its performance. If the ground truth specifies that image A should be returned when querying with B, then failing to identify the unique descriptor of A among the near neighbors of B immediately negatively impacts quality. This is to contrast with the case where multiple local descriptors are used to search the images that are similar to a single query image.

In this case, each image yields many descriptors (several hundreds for high-quality images), where each descriptor describes a small "local" area of the image. To retrieve the images that are similar to a query image, a k-nn search is run for each local descriptor computed on the query image. Each nearest neighbor found is voting for the image it is associated with. Then, the most similar images are found by ranking the images according to their number of votes.

This process is in fact based on redundant information—the more votes one specific image gets, the more similar it is. For that reason, missing some near neighbors does not matter that much, as there are many descriptors. Indeed, with local descriptors, quality is negatively impacted if close to no descriptors of the ground truth image A are found when querying with all the descriptors of image B; in this case, A receives close to no votes. This, overall, dramatically releases the pressure on the performance of the index; typically, having as few as a dozen of votes for A (or about 1%) is enough to have A in the list of the images found similar to B. Then, the quality of searches at the image level is preserved, even if some individual descriptors were lost during the process.

We have defined two families of benchmarks. The first one is dedicated to evaluate the performance of indexing schemes when running single descriptor queries. The second one evaluates the performance of indexing schemes at the level of image recognition. We describe both benchmarks in the following sections.

II.3. Ground Truth for Benchmarking Single Descriptor Queries

This section defines a benchmark at the level of individual descriptors; it is intended to directly reflect what happens in the feature space. In the literature, one of two different approaches is typically used to define the ground truth set. The first approach is to run an exact k-nn neighbor search for every query descriptor, leading to a result set of fixed cardinality, but with arbitrary distances. The second approach is to run an exact ε-range search for each query descriptor, which returns all neighbors within distance of ε from the query point, leading to a result set with a bounded distance, but of arbitrary cardinality. In both cases, an exhaustive sequential scan is typically used to ensure that the result lists defining the ground truth truly reflect the contents of the descriptor collection. The result quality of the indexing scheme in question can then be computed by comparing its results to these ground truth sets. Of course, both methods are highly sensitive to the choice of k or ε, respectively.

Results published by Beyer et al. [Beyer et al., 1999] as well as the ones by Shaft and Ramakrishnan [Shaft and Ramakrishnan, 2006], however, have shown that high-dimensional data sets must exhibit some *contrast* to be indexable and to draw any meaningful conclusion from search results. In their work, contrast means that a nearest neighbor must be significantly closer to the query point than most other points in the dataset in order to be considered meaningful. In the absence of contrast, collections suffer from vanishing variance and instability of near neighbors, which preclude the construction of meaningful result sets. A direct consequence of the theoretical analysis of [Shaft and Ramakrishnan, 2006] is that it is possible to construct a contrast-based ground truth set, against which indexing schemes can be compared.

In order to construct such a set, a sequential scan may be used to determine, for each query descriptor, all the descriptors in the dataset that fulfill a given contrast criteria. Using a contrast-based ground truth set has several theoretical benefits. First, the size of the ground truth set tends to be small compared to the k-nn approach, which collects (irrelevant) neighbors regardless of their distance from the query descriptor. Second, using the contrast-based ground truth alleviates the two typical problems that ε-range search faces. On one hand, when query points fall in very dense areas, a very large number of vectors are typically returned, although the results are hardly distinguishable one another. On the other hand, when query points fall in sparse areas, an ε-range query may return no result, while there may be many useful answers in the collection. Overall, therefore, building a ground truth based on contrast will allow more reliable result quality measurements.

According to Lowe, computing SIFT over an image collection produces a contrasted set of descriptors [Lowe, 2004]. In his work, Lowe considered the nearest neighbor n_1 of a query descriptor q meaningful if and only if $d(n_2, q)/d(n_1, q) > 1.8$, where n_2 is the second nearest neighbor. When the nearest neighbor passed the criteria threshold, then further checks were run to see whether n_1 was indeed a modified copy of the query descriptor. If the nearest neighbor did not pass the criteria threshold, then n_1 was rejected and no answer returned. Since, for many applications, a query may have more than one meaningful result, we adapt Lowe's criterion, by saying that returned neighbor n_i is meaningful with respect to contrast c (default value of c is 1.8) when $d(n_{100}, q)/d(n_i, q) > c$. In fact, we can generalize Lowe's criterion, by saying that returned neighbor n_i is meaningful with respect to contrast c when $d(n_j, q)/d(n_i, q) > c$, where $j \geq 2$ and $i < j$. In our work we have found, however, that with j between 2 and 100, the number of descriptors passing the contrast criterion grows fast, while for $j > 100$ it grows slowly. We have therefore used $j = 100$ in the remainder.

We have conducted a detail analysis of the quality results obtained in order to determine which of those three ground truth generation schemes works the best. We are particularly interested in checking the *recall* observed when using each ground truth.

II.3.1. Dataset, Queries and Matches

We used a set of 179,443,881 128-dimensional SIFT descriptors that was obtained by extracting local features from an archive of about 150,000 high-quality press photos. We then extracted 500,000 query descriptors from transformed versions of images from this collection. These transformations are created using the Stirmark benchmarking tool [Petitcolas et al., 1998, Petitcolas et al., 2001] and consist, among others, of rotation, rescaling, cropping, affine distortions and convolution filters. It has been shown that SIFT descriptors cope very well with most of these distortions, meaning that a high percentage of corresponding descriptors are close in the high-dimensional space. We have then run a sequential scan to calculate the 1,000 nearest neighbors for each query descriptor, yielding 500 million neighbors in all.

Note that the semantics of the copyright protection application, from which the workload is drawn, is such that for each query descriptor precisely one descriptor in the collection is a *correct match*, while the remainder should be considered *false matches*. In our collection, a total of 248,852 query descriptors found a correct match among the top 1,000 neighbors, or slightly less than 50%. While this may at first seem a low percentage, it is still a good recognition performance considering that some query descriptors originated from severely modified images.

We then generated three different versions of a ground truth, respectively according to a k-nn approach, a ε-distance approach and a *contrast*-based approach. Our goal is to determine which ground truth is the one that includes a large number of the 248,852 correct matches and only a

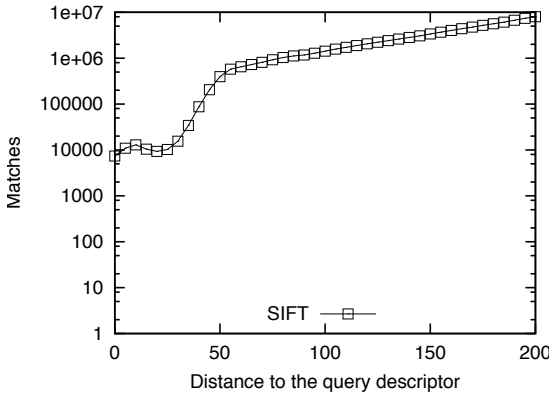


Figure II.1.: Distribution of all neighbors based on distance to the query descriptor

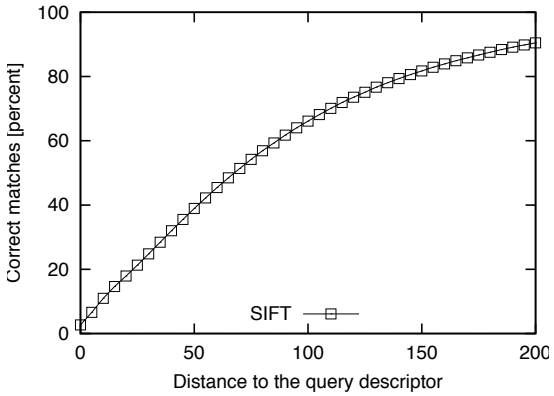


Figure II.2.: Cumulative distribution of correct matches based on distance to the query descriptor

small number of false matches.

II.3.2. Ground Truth Based on k-nn

When taking a close look at the individual results we observed that the correct matches that appeared among the 1,000 nearest neighbors were in most cases ranked first in the result set. This indicates that by far the best choice for building a ground truth based on k nearest neighbors, would be by choosing $k = 1$. But even with $k = 1$, however, more than half of the neighbors in the ground truth set would be false matches. Furthermore, for many other applications, choosing a ground truth set of $k = 1$ would be too restrictive.

II.3.3. Ground Truth Based on ε-Distance

We now analyze how the absolute distance between the query descriptor and returned neighbors affects the result quality. Figure II.1 shows the distribution of all 500 million neighbors depending on the distance of each neighbor to the query descriptor. The x-axis shows the absolute distance (corresponding to varying ε), while the y-axis shows the number of neighbors with approximately that distance (the point at 0 corresponds to a distance of 0, while the point at 5 corresponds to the distance range $(0, 5]$, and so on). We observe that the number of descriptors stays rather uniform and small for short distances. Once the distance surpasses 25, however, we can see an exponential increase in the number of neighbors at each distance range (note the logarithmic scale). Recall that in our application almost all of these descriptors are false matches.

Figure II.2 shows the cumulative distance distribution of the correct matches; the x-axis is the distance from the correct match to the query descriptor, while the y-axis shows the cumulative fraction of correct matches found below that distance. From the figure we see that about two thirds of the correct matches can be found within an ε-distance of 100, and that within this distance they are rather uniformly distributed. The final third lies beyond a distance of 100, where the likelihood of finding further neighbors slowly becomes smaller; the last correct matches can actually be found at a distance of 370.

More importantly, however, Figure II.2 shows that fewer than 20% of the correct matches are found at a distance smaller than 25, which is where the number of false matches started

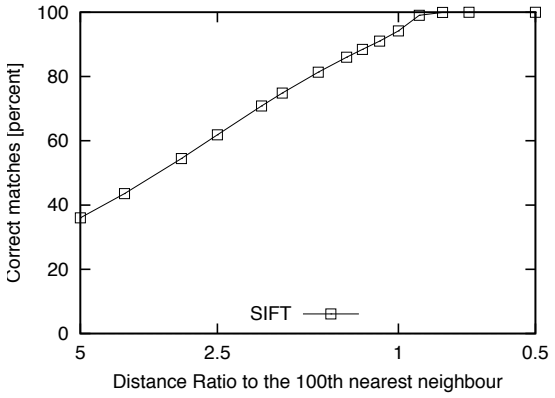


Figure II.3.: The cumulative distribution of correct matches based on contrast

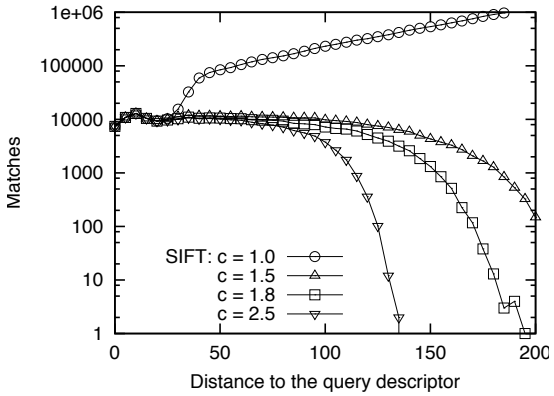


Figure II.4.: Distribution of neighbors passing the contrast criterion by distance to query descriptor, for various contrast thresholds

increasing exponentially. Thus, it is impossible to select a global ε for building a ground truth set which includes a large number of correct matches and only a small number of false matches.

II.3.4. Ground Truth Based on Contrast

Finally, we consider the effect of contrast on the quality of the ground truth set. Figure II.3 shows an analysis of the correct matches based on different thresholds of the contrast criterion. The x-axis shows the contrast c, while the y-axis shows the percentage of correct matches with contrast higher than c, as $d(n_{100}, q)/d(n_i, q) > c$. The figure shows that 36% of the correct matches are more than five times closer in distance than the 100^{th} nearest neighbor in the result list. For $c = 1.8$, which is the value that Lowe recommended, 186,290 out of 248,852 correct matches, or about 74.9%, pass the contrast threshold. About 20% of the correct matches have a contrast threshold lower than 1.5, and are therefore rather hard to detect from the false matches.[1]

Figure II.4 shows the effects of the contrast criterion on the number of descriptors that pass the threshold filter (these include the correct matches). This time, the x-axis shows the absolute distance from the result descriptor to the query descriptor, while the y-axis shows the number of descriptors found at each distance. Overall, we observe that a contrast threshold of $c = 1$ shows an exponential increase in the number of descriptors (similar to Figure II.1, but at a smaller scale since at most 100 neighbors are considered), while all values of $c \geq 1.5$ avoid this behavior and show a well controlled number of descriptors; the higher the threshold, the fewer descriptors are returned.

Comparing Figures II.3 and II.4, we see that choosing a higher contrast threshold gives lower recall but fewer false matches, and vice versa. Comparing these to Figures II.1 and II.2 shows that any choice from 1.5 to 2.5 performs very well compared to the ε-based criterion. So the threshold of 1.8, proposed by Lowe, seems reasonable.

With the threshold $c = 1.8$, a total of 248,212 descriptors pass the contrast filter.[2] As described above, the number of descriptors that are both correct matches and pass the $c = 1.8$ contrast

[1] A small portion of the correct matches has contrast smaller than 1, which means that they were found at a rank higher than 100.

[2] The fact that this number is similar to the number of correct matches in the sequential scan results is purely

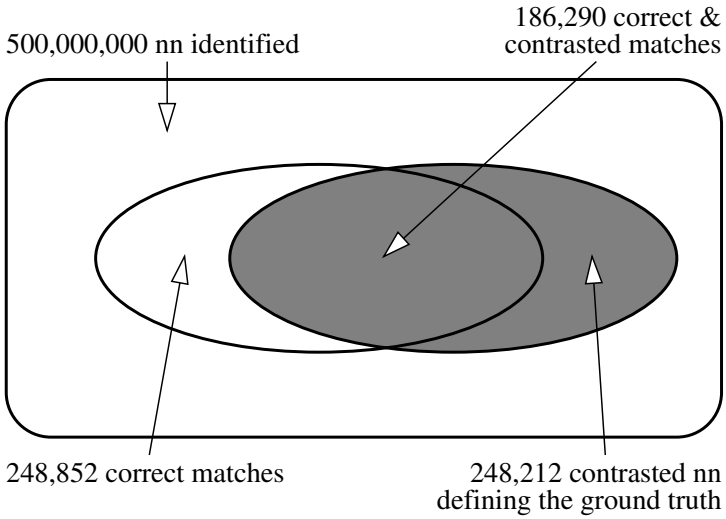


Figure II.5.: Illustrating the Ground Truth Based on Contrast

criterion is 186,290. Thus, about 75.1% of the descriptors in the contrast-based ground truth set are correct matches and about 24.9% are false matches.

Figure II.5 illustrates the ground truth based on contrast. A sequential scan identified 500M neighbors from 500k query points. Only 248,852 are correct matches (see Section II.3.1). Regardless of these matches, only 248,212 neighbors are contrasted enough from the 500M that were identified (see above). This is defining the ground truth (the grey area on the figure). This ground truth is meaningful from the point of view of the application because 75.1% or 186,290 of the points in the ground truth are correct matches.

II.3.5. Discussion

The discussion above shows that using a contrast-based criterion to construct the ground truth set is clearly preferable to using either k-nn or ε-distance, as using the contrast-based criterion yields the best ratio between correct matches and false matches (about 3:1 for $c = 1.8$). Furthermore, it is the only approach with solid theoretical underpinnings. As a result, we use contrast-based ground truth sets in the remainder of this manuscript when evaluating the quality of single descriptors queries. We typically use $c = 1.8$ to build the ground truth set.

When evaluating systems, this ground truth will be used as follow. The system will be probed with the 500k queries, each returning 1,000nn. Then, a contrast-based filter will be applied to these 500M returned neighbors. The neighbors passing the filter (because they are contrasted enough) will be compared to the ones belonging to the ground truth. From this comparison, a measure of the recall can easily be computed. This evaluation protocol is for example used later in this manuscript, see Section III.4.7 page 42.

II.4. Image Dataset and Ground Truth

This section describes a benchmark that is defined to evaluate the quality of a CBIR system when identifying images that are similar to a query image. In order to evaluate the performance of in-

by coincidence.

Family	Nature of Transformation	Stirmark Parameters	Image Suffix
Affine	X-shearing, Y-shearing, XY-shearing	3*6 parameters matrix. No translation	AFFINE_1, ...
Convolution	3x3 mask	Gaussian Filtering Sharpening	CONV_1, CONV_2
Crop	Removes x% of the surface of the image	x=50%, 75%	CROP_50, CROP_75
Jpeg	Compression. Quality levels=	15, 80	JPEG_15, JPEG_80
Median Filter	Median Filtering	3x3 mask	MEDIAN_9
Noise	Randomly picks x% of the pixels. Change their value randomly.	x=5%	NOISE_5
PSNR	Add to all pixels parameter value/4. Decide on sign randomly	50	PSNR_50
Rescaling	Rescales image. Ratio in %.	50 %, 75 %, 90 %, 110 %, 150 %, 200 %	RESC_50, ...
Line Removal	Removes 1 line of pixels every x pixels	x=10, 35, 60, 85	RML_10, ...
Rotation	Rotation by x degrees. Creates 4 black triangles around image.	x=-2, -1, 1, 5, 15, 30, 45, 90	ROT_-2, ROT_-1, ROT_1, ...
Rotation+Crop	**Rotation**. Auto crops image to entirely remove black triangles.	same x	ROTCROP_-2, ...
Rotation+Scale	**Rotation+Crop**. Auto rescales the cropped image to its original size.	same x	ROTSCALE_-2, ...
Self similarity	Spatial swapping of blocks	Hue component (hsv) Blue channel (rgb) Y component (yuv)	SS1, SS2, SS3

Table II.1.: Contents of the 49k image benchmark

dexing when searching for similar images, we have adopted the traditional quasi-copy paradigm. We apply a fixed number of predefined image transformations to a particular collection of images. We then drown the original images used to create the transformed quasi-copies into a larger random image collection playing the role of distracting contents and we use the transformed images as queries, searching for the original ones. We then check if the original images are found, and when they are found, we record their ranks in the final result lists.

Most of our experiments searching for similar images use what we call the "49k image benchmark". Some also use the "Copydays image benchmark". Both are described in detail now.

II.4.1. 49k Image Benchmark

We randomly picked 1,000 images from Flickr that are very diverse in contents. Each image was then given to Stirmark software [Petitcolas et al., 2001, Petitcolas et al., 1998] in order to generate 49 transformations, reported in the Table II.1. Overall, this generates 49,000 quasi-copy query images. Note some of these quasi-copies are quite dissimilar to their original counterpart. For example, the CONV_2 transform tends to be extremely dark, and very few SIFT descriptors can be computed from that contents; this quasi-copy is very hard to find. The MEDIAN_9, NOISE_5 and PSNR_50 are also quite different, making the identification of their original counterparts very challenging, especially because the SIFT descriptors computed on the quasi-copies are either at different scales, either at very different locations in the images because the visual noise produces very different local differences of Gaussian extremum. Finally, it is worth noting crops are inherently challenging since they dramatically reduce the number of SIFT that can be computed on the quasi-copies, which in turn strongly decreases the number of possible matches.

Family	Nature of Transformation	Parameters	#
Crop	Removes x% of the image surface	x=10, 15, 20, 30, 40, 50, 60, 70, 80	1,413
Scale+Jpeg	Quasi-copy has 1/16 pixels. Then Jpeg compression, quality=	 3, 5, 8, 10, 15, 20, 30, 50, 75	1,413
Manual	printed then scanned, paint, severe blur, screenshot		229

Table II.2.: Contents of the Copydays image benchmark

II.4.2. Copydays Image Benchmark

The Copydays dataset has been created by Jégou and Douze. It is publicly available and had first been used in [Douze et al., 2009]. We are using this image dataset for our experiments (see next chapter) because it contains quasi-copies images that are more severely distorted than the ones in the "49k" image set. The original images are hence much harder to find from their quasi-copies. Copydays contains 157 original images. Three families of transformations have been applied, resulting in a total number of quasi-copies equal to 3,055. This is summarized in the Table II.2. The 229 manual transformations are particularly hard since they generate quasi-copies that are visually extremely different from their original counterparts. For example, Figure II.6 show two original images, (a) and (c) and one of their strong variant, respectively (b) and (d). Please note that the sizes of the originals and their variants is preserved in this figure.

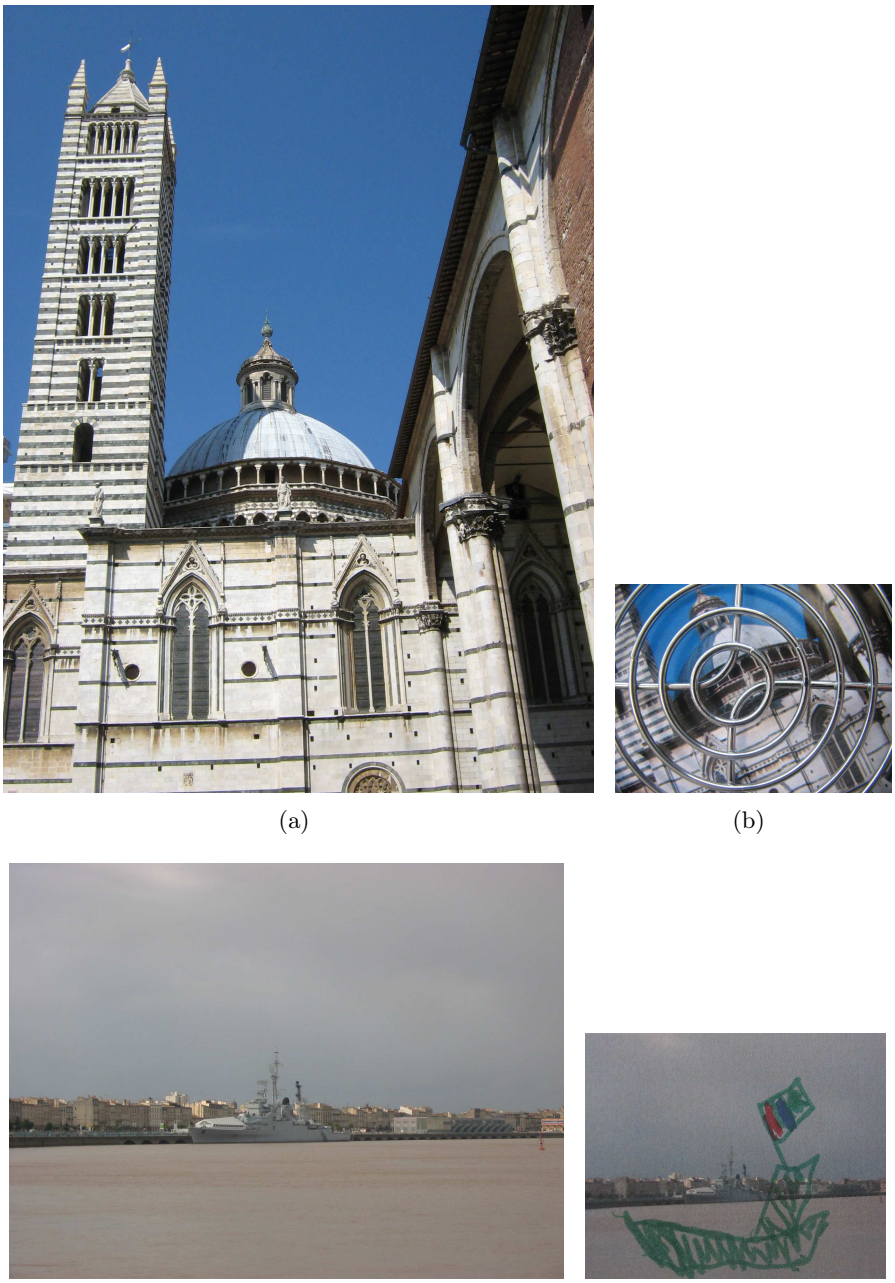
II.4.3. Distracting Image Collections

In order to evaluate the quality results produced by indexing schemes, we typically drown the single descriptor ground-truth (Section II.3.4), the 49k image set and the Copydays image set into a larger set of images randomly obtained from the Web. We are using two different collections of distractor images, differing essentially by the resolution of the images.

The "512 Pixels" Distracting Images Set

The first distracting set contains pictures randomly downloaded from Flickr between 2009 and 2011. We have gathered almost 30 million such images. All pictures have been resized such that their longer edge is 512 pixels long. We did this to somehow limit the number of SIFT computed over each image to an average of 1,000. All the original images used to create the quasi-copies of the 49k and Copydays image benchmarks were resized accordingly. Queries and their counterparts are therefore consistent with respect to the distracting images into which they are drown.

We then varied the number of distracting images in order to study its impact on the result quality of the indexing schemes. We therefore created 6 different data collections, all including the single descriptor ground truth, the original images from the 49k image set and from the Copydays image set, but having a different number of distracting images. These data collections contain roughly 30 million, 180 million, 300 million, 2.5 billion, 3 billion and 30 billion SIFT descriptors. Their names (used all over in this manuscript) and precise specifications are given by the Table II.3.



(a) (b)

(c) (d)

Figure II.6.: Four examples from Copydays: (a) and (c) are two original images; (b) and (d) are two strong variants used as queries. Size ratio is preserved

Set Name	# of SIFT Descriptors	Size on disks	# of images
30M	28,799,690	3.8G	26,583
180M	179,443,881	23.6G	179,277
300M	305,443,749	40.3G	334,268
2.5G	2,485,568,191	328G	2,259,277
3G	3,040,856,472	401G	2,970,596
30G	28,484,904,924	3.7T	28,969,271

Table II.3.: Figures for the image collections used in this manuscript. Resolution= 512 pixels

The "150 Pixels" Distracting Images Set

The second distracting set of images we used in our experiments has been created for the Quaero project.[3] One of the Quaero partners, Exalead, collected roughly 100 million images by harvesting the Web. To limit the size of data and to facilitate sharing among the partners in the Quaero project, images have been resized to only 150 pixels on their largest side. SIFT descriptors were then extracted from these images, resulting in about 30 billion descriptors, i.e. 300 SIFT descriptors per image on average. We also created subsets from this distracting image collection. This set is called the "100M image set".

Here, again, the "49k" and the "Copydays" sets have been resized before drowning the original images in the distractors.

[3]Quaero is a research and innovation program addressing automatic processing of multimedia and multilingual content.

Chapter III

Scalar Quantization Based Indexing

Contents

There is a very large body of work concerning high-dimensional indexing techniques relying on scalar quantization operators. Scalar quantizers are simple to design, to implement, to control. Scalar quantizers have played a major role in facilitating giant progress in the state-of-the-art of high-dimensional indexing techniques. In particular, they are at the very core of probably one of the most popular technique used today, namely LSH [Gionis et al., 1999]. Scalar quantization techniques have been applied to many problems that fundamentally need similarity search operations. They have limited performance however, as it is discussed in this chapter.

This chapter describes our experience with working with scalar quantization based indexing techniques. Working with scalar quantization schemes, implementing, evaluating and understanding them deeply, using them to confirm or contradict what our intuition says in front of the complicated curse of dimensionality phenomenon kept us busy for about a decade. This chapter, however, only describes some elements that are at the surface of the work we did.

A lot of this work has been done in very tight cooperation with other researchers having a strong background in databases. For this reason, the analysis we did during this decade includes typical database points of view. This essentially means secondary storage is by default required to cope with extremely large scale data collections as well as to offer persistent storage to indexed data.

This chapter is structured as follows. We first introduce the general principles of scalar quantization based indexing techniques in Section III.1. We then proceed by reviewing the two scalar quantization schemes that had the most influence on our work. We first describe LSH in Section III.2 and then Omedrank in Section III.3. We detail why each was so influential as well as the severe problems each one is facing. The lessons drawn from criticizing these two schemes allowed us to design a specific indexing scheme that uses scalar quantization. It is the NV-Tree, and it is described in details in Section III.4.

The NV-Tree is dedicated to managing very large collections of high-dimensional descriptors on disks. It copes with concurrent updates and it is resistant to failures. It is probably our most achieved contribution from a decade of work. The technology at the core of the NV-Tree has been transferred to a start-up company having now a dozen of employees and it is used for digital forensic application by several anti-terrorist police forces.

Last, Section III.5 concludes this chapter.

III.1. Introduction

Quantization is the process of mapping a large set of input values to a smaller set of output values [Gray and Neuhoff, 1998]. It is a lossy process as some information is lost during this many-to-few mapping operation. Quantization is typically for compressing signals: it maps the continuous values of a signal over a discrete time line into a series of discrete values over a discrete time line.

Scalar quantization is certainly the most common type of quantization. It is typically noted $\hat{x} = q(x)$, and the function $q()$ maps one input value x into a scalar (one-dimensional) value $\hat{x}$. Historically, $x \in \mathbb{R}$.

More formally, a scalar quantizer is an application q from $\mathbb{R}$ to a discrete and possibly infinite but countable set of scalar, mono-dimensional values defining $\mathcal{F}$:

$$\mathcal{F} = \{\hat{x}_1, \hat{x}_2, \ldots\}, q : \mathbb{R} \rightarrow \mathcal{F}$$

$\mathcal{F}$ is the quantification dictionary, also called the code-book. There are many ways to define the values $\hat{x}_i$ in $\mathcal{F}$ as it is described in [Gray and Neuhoff, 1998]. Two simple examples: All the intervals between $\hat{x}_i$ and $\hat{x}_{i+1}$ have the same length with the uniform scalar quantizer; dead-zone quantization schemes have more intervals around the values of x that are close to zero, ...

The number of possible items in $\mathcal{F}$, their actual values, the way each value is encoded, the way their distribution covers the input space as well as the possible input value clipping all together define the properties of the quantization scheme.

Overall, quantization is used because it turns close enough input values into one representative output value. Values that are sufficiently different receive a different representative output value. This process can be seen as a form of cell construction, each cell being identified by one of the $\hat{x}_i$ output value, and each cell capturing several input values that are close enough to be mapped to the same output value. This is somehow achieving the very same goal as indexing, and this observation motivated researchers to apply quantization techniques to high-dimensional input values.

When using scalar quantization in the context of high-dimensional indexing, then $x \in \mathbb{R}^d$, but the definition of $\mathcal{F}$ does not change. It is still defined by a set of scalar values representing the high-dimensional vectors. The most frequent approach to implement q is to use *orthogonal projections* onto a line. Orthogonal projections turn high-dimensional values into scalar values that are the offsets of the projections with respect to the origin. Such orthogonal projections map high-dimensional (input) values into mono-dimensional (output) values. There are possibly many strategies for defining the projection lines, from totally random to carefully picked lines.

The typical usage of scalar quantization schemes for indexing is as follows, using a very simplified view, however. The collection of descriptors to index is first projected onto one line, and what results from the projections is kept in one data structure. With projections, it is likely that two descriptors that are close in space will have their projected values close on the line. Descriptors that are far apart in space are likely to have their projected value quite different on the line. It is also possible that some descriptors that are far away in space turn out to have their projected values close on the line. This process constitutes the indexing phase of the collection, and it is typically performed off-line. When completed, then searching is possible and proceeds as follows. The query descriptor given by the user is projected onto the same line as the one used at indexing time. This results into generating one projected value. Then, the descriptors from the indexed collection that have their projected value close to the one of the projected query descriptor are scrutinized to possibly be part of the final result that will be returned to the user.

Several high-dimensional indexing schemes use some form of orthogonal projections. We now move to describing LSH and Omedrank that are, from our point of view, two scalar quantization based indexing schemes that had the most influence on our understanding of the field. Their properties, their strengths as well as their weaknesses enabled us to design the NV-Tree, our central contribution in this domain.

III.2. Locality-Sensitive Hashing—LSH

Locality Sensitive Hashing (LSH) by Indyk et al. [Gionis et al., 1999, Shakhnarovich et al., 2006] is based on the concept of projecting descriptors onto a set of random lines and quantizing the locations along each line into buckets. This is hashing descriptors. Hash functions are constructed such that descriptors that are close in the feature space (hence, similar) are likely to collide into the same buckets with high probability, while dissimilar descriptors are likely to end-up in different buckets.

When constructing the index, each descriptor of the data collection is inserted into $\mathcal{L}$ hashing buckets determined after having applied $\mathcal{L}$ hash functions to its multi-dimensional components. At query time, the query descriptor is also hashed $\mathcal{L}$ times, which in turn determines $\mathcal{L}$ buckets. For all candidate descriptors stored in these buckets, the LSH algorithm computes the exact distance to the query point, eventually producing the set of k neighbors.

LSH was first published for the binary Hamming space [Gionis et al., 1999] and then later extended to the L_p norm [Datar et al., 2004]. LSH is also at the root of many papers building on its principle, trying to fix some of its problems in terms of quality and/or complexity.

We now turn to presenting more details on LSH before moving to discussing some of its key problems and major extensions.

III.2.1. Indexing and Searching with LSH

This section gives a more formal definition of the Euclidean version of LSH. Overall, indexing with LSH follows three steps. Searching is described later. First, high-dimensional descriptors are turned into hashed values via the use of random projections. Second, these hashed values are randomly concatenated, giving $\mathcal{L}$ new hashed values. Third, these $\mathcal{L}$ hashed values are mapped to $\mathcal{L}$ bucket numbers, buckets storing the indexed descriptors. These three steps are detailed below.

Let $\mathcal{C}$ be a collection of n descriptors lying in $\mathbb{R}^d$. Let $d(.,.)$ be the function computing the Euclidean distance between two descriptors.

Indexing $\mathcal{C}$ with LSH requires to project $\mathcal{C}$ onto a limited set of m directions characterized by the set of vectors $a_{i,1\leq i\leq m} \in \mathbb{R}^d$. Each direction a_i is drawn from an isotropic random generator.

To index a given vector $x \in \mathcal{C}$ with LSH, x is first hashed into an integer value:

$$h_i(x) = \left\lfloor \frac{\langle x|a_i\rangle - b_i}{w} \right\rfloor$$

where w is the quantization step chosen according to the data (see [Shakhnarovich et al., 2006]), the offset b_i is uniformly generated in the interval $[0, w[$ and where the inner product $\langle x|a_i\rangle$ is the projected value of the vector x onto the ith direction.

The family of hash functions $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{Z}\}$ is called $(r, \lambda r, p_1, p_2)$-sensitive for $d(.,.)$ if, for any $q \in \mathbb{R}^d$, $p \in \mathcal{C}$ and $i \in [1, m]$:

$$\begin{aligned} P(h_i(q) = h_i(p)) &\geq p_1 \quad \text{if} \quad d(q,p) \leq r \\ P(h_i(q) = h_i(p)) &\leq p_2 \quad \text{if} \quad d(q,p) \geq \lambda r \end{aligned}$$

with $\lambda > 1$ and $p_1 > p_2$.

The intuition behind these settings is that descriptors that are close enough, within the distance r, are more likely (this is p_1) to be hashed to the same value than descriptors that are far apart (those within a distance greater than λr, with at most probability p_2).

While $\mathcal{H}$ is defined here for the L_2 metrics, other definitions of $\mathcal{H}$ have been established when $d(.,.)$ is the Jaccard measure, when it is the Hamming distance, as well as when $d(.,.)$ is the L_1 metrics [Indyk and Motwani, 1998]. More broadly, Datar et al. have defined $\mathcal{H}$ for various L_p norms based on p-stable distributions [Datar et al., 2004].

A single hash function from $\mathcal{H}$ is not discriminative enough by itself because only one direction among d is used to partition the vectors. Therefore, a second level of hash functions, also based on the family $\mathcal{H}$, is subsequently defined. This level is formed by a family of $\mathcal{L}$ functions constructed by concatenating several functions from $\mathcal{H}$. Hence, each function $g_j, j \in [1, \mathcal{L}]$ of this family is defined as

$$g_j = (h_{j,1}, \ldots, h_{j,k}),$$

where the functions $h_{j,i}$ are randomly chosen from the set $\mathcal{H}$ of hash functions. At this point, a vector x is temporarily indexed by a set of $\mathcal{L}$ vector of integers $g_j(x) = (h_{j,1}(x), \cdots, h_{j,k}(x)) \in \mathbb{Z}^k$.

Theoretically, the $\mathcal{L}$ $g_j(x)$ values are sufficient for constructing the index. However, since $\mathbb{Z}^k$ is very large, it is not practical to use $g_j(x)$ as a bucket number into which inserting x, for all $g_j, j \in [1, \mathcal{L}]$. Directly using the integers $g_j(x)$ as bucket numbers creates a very large number of buckets, most of them remaining empty even after hashing a large number of descriptors.

LSH proposes to use a universal hash function u_1 to obtain from $g_j(x)$ an integer lying in the interval $[0, c[$. This mechanism turns the theoretical $\mathcal{L}$ bucket numbers lying in $\mathbb{Z}^k$ into

$\mathcal{L}$ numbers, each ranging in $[0, c[$. The value of c must be high enough to avoid, with high probability, the collision of vectors x and t that are dissimilar enough in the feature space. For the vector of integers $g_j(x)$, the function u_1 is defined as

$$u_1(g_j(x)) = \left(\left(\sum_{i=1}^{k} r'_i h_{j,i}(x) \right) \mod P \right) \mod c,$$

where P is a prime (e.g., $P = 2^{32} - 5$) and the various values for r'_i are random integers. The integer $u_1(g_j(x))$ designates a bucket in the hash table associated with the hash function g_j. This process has to be repeated $\mathcal{L}$ times, after which x is inserted into $\mathcal{L}$ buckets, each being determined by the number $u_1(g_j(x))$ when j varies from 1 to $\mathcal{L}$.

To further reduce the probability of undesired collisions (e.g., if c is chosen small to reduce the memory usage), a second hash function u_2 similar to u_1 can be used:

$$u_2(g_j(x)) = \left(\sum_{i=1}^{k} r''_i h_{j,i}(x) \right) \mod P,$$

where the r''_i are random integers different from the r'_i. In that case, the $\mathcal{L}$ integers $u_2(g_j(x))$ are stored together with the vector identifier of x in the $\mathcal{L}$ buckets numbered $u_1(g_j(x))$ when j varies from 1 to $\mathcal{L}$. The $\mathcal{L}$ $u_2(g_j(x))$ values are used as an additional filtering criterion.

At query time, the query vector q is projected onto each of the $\mathcal{L}$ random lines, producing a set of $\mathcal{L}$ integer vectors $\{g_1(q), \cdots, g_{\mathcal{L}}(q)\}$. This subsequently produces $\mathcal{L}$ bucket numbers thanks to the application of the u_1 functions, each ranging in $[0, c[$. The values produced by the second hash function u_2 are also determined for the query vector. The exact distance between q and all the descriptors that have identical u_2 values and that are stored in the union of these $\mathcal{L}$ buckets are computed. Then the smallest k distances are used to return to the user the corresponding descriptors from the indexed collection.

III.2.2. Problems with LSH

Projections Perform Poorly

LSH does the hashing of the vectors by projecting them onto random lines. This works best when the projection lines are orthogonal. Projection lines that are (quasi) co-linear order (quasi) identically the points on the different lines, which is redundant and thus brings useless information. To best work, projections should spread as much as possible the projection values along lines. This is the rationale for sometimes preferring not using *random* lines but lines determined after having ran a principal component analysis (PCA) on data. The resulting lines are the ones that best spread the data.

In high-dimensional spaces, projections (either random or PCA driven) turn out to perform poorly. The projections values are not at all well spread but tend to be concentrated inside a very narrow interval, around zero. This means that most vectors are indeed found to be quasi orthogonal to any projection line.

We conducted a small experiment to illustrate this behavior, in practice. We used a collection of about 300 million SIFT descriptors extracted from roughly 330k real world images randomly downloaded from Flickr.[1] We then randomly created a set of 20,000 random lines. We projected 100,000 vectors randomly picked from this data collection onto these lines. We then determined

[1] This corresponds to the third line of Table II.3 from Chapter II, page 25.

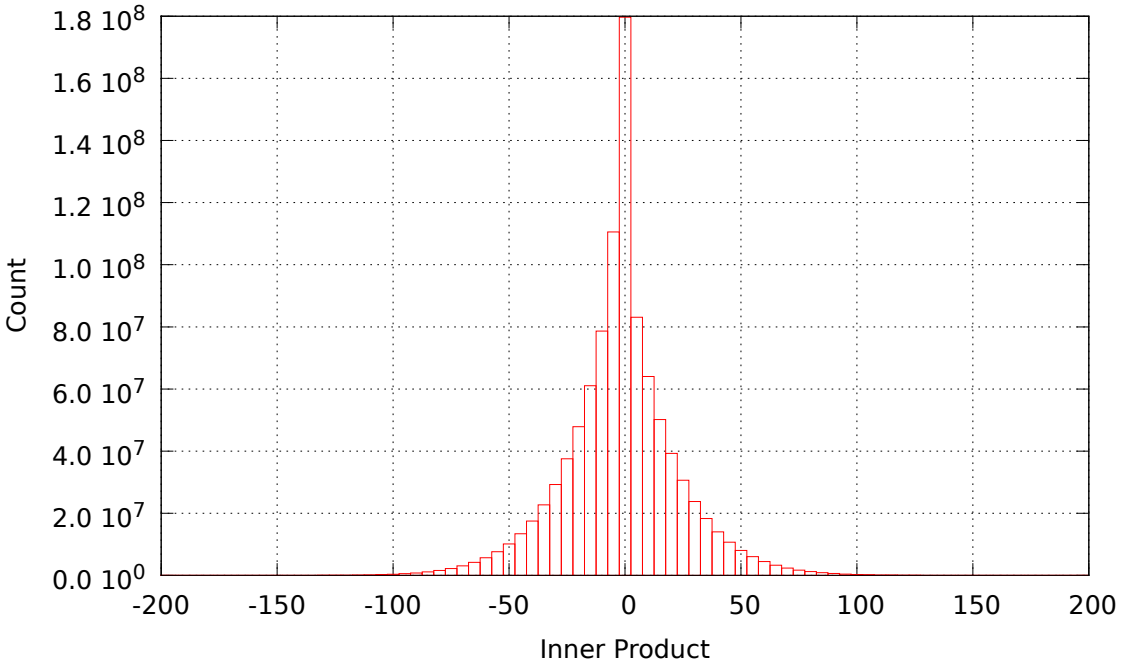


Figure III.1.: Distribution of inner products. 100,000 vectors against 10,000 lines. Real data. Dimension=128

the 10,000 lines on which the projection values have the largest variance. We subsequently quantized the projection values obtained using these 10,000 lines into bins and counted the number of times the projection values fall in each bin for all the lines. The result of this experiment is given by Figure III.1. Please note that SIFT vectors have all the same norm. This Figure shows the distribution of the 10^9 inner products performed during this experiment. It shows that roughly 18% of the vectors are almost orthogonal, that about 37% are in the three central bins, closest to orthogonality and that 52% are in the 5 central bins. We repeated that same experiment with purely random quasi-orthogonal lines. A very similar plot is obtained.

Projections are not able to nicely spread the data, even projections better fitting the data than what random lines could do. Projected values are clearly very skewed. This phenomenon is at the roots of two major problems LSH has to face, and which are detailed next.

Many Hash Functions are Needed

The first immediate consequence of the poor performance of projections forces LSH to require using a fairly large number of hash functions (this is the $\mathcal{L}$ parameter, see above) to capture the proximity of the descriptors in the feature space. For example, more than 100 hash tables are used in [Gionis et al., 1999] and 583 are used in [Buhler, 2001]. That large number has two severe consequences. First, the computation cost of applying so many hash functions slows down the process, this being particularly problematic at index creation time since the number of points to process is gigantic at large scale. Second, there exist as many copies of the entire data collection as there are hash tables. When collections are terabyte sized and when hundreds of tables are needed, storage becomes an issue. It is likely such storage requirements will exceed the size of the main memory, forcing LSH to lookup data on disks, which is a costly process. LSH does not include any specific mechanism to friendly deal with I/Os.

Buckets are Severely Imbalanced

LSH uses hashing, and therefore it gives best results when hashed data is evenly distributed in the buckets. Unfortunately, this is not at all the case for high-dimensional data as shown earlier. Here, data tend to follow a normal distribution and hashing therefore tends to create buckets having very different cardinalities. This has a severe impact on the observed response time of LSH: it is completely unpredictable since the buckets needed to answer a particular query might be very large and/or very small, which changes the number of distances to compute as well as the numbers of disk reads to perform when fetching buckets. In [Lejsek et al., 2009], we observed for some specific LSH settings that some buckets were orders of magnitude larger than others—performance suffers.

Fetching Buckets from Disks is Costly

Once the buckets have been determined for a particular query, then all the descriptors in these buckets need to be used in distance calculations. This is particularly costly at large scale for two reasons. First and mainly, the scale of the collection together with the compulsory use of many hash tables cause buckets to be stored on disks. Fetching buckets from disks into memory is inherently costly, with the highest possible cost since disk reads are random. Second, because the size of the buckets varies a lot in an unpredictable way, there is no way to optimize the accesses to disks. It is likely some hash buckets will be much smaller than an I/O granule, and in this case more data than the desired amount is fetched in memory for pure waste when requesting such a bucket; it is likely some other buckets will span many I/O granules, dramatically augmenting the disk reading costs.

Computing Distances is Costly

Once in memory, the descriptors kept in these buckets are used for computing many distances. This puts high pressure on the CPU. Overall, LSH suffers from a I/O bottleneck caused by the many hash tables; it also suffers from a CPU bottleneck caused by the many distances to compute.

III.2.3. Extensions

Many researchers have tried to improve the behavior of LSH. Observing that getting high-quality results require the costly use of many hash tables, Panigrahy proposes in [Panigrahy, 2006] to use only few tables but to analyze more than one bucket per table. [Lv et al., 2007] builds on this basic idea and gives a more efficient scheme. Overall, these schemes trade time for space requirements. They basically determine the likelihood of buckets to store descriptors that could improve the query results—this series of works are known as Multi-probe LSH schemes. Better ways to estimate this likelihood have been proposed [Joly and Buisson, 2008] as well as better ways to pick random lines for projections [Zhang et al., 2010b].

Other works investigate a radically different path to improve the behavior of LSH. They basically propose to replace the hash functions that perform scalar quantization with vectorial quantizers that better preserve the locality of data in space. Providing details on vectorial quantization makes the bulk of the next chapter.

III.3. Omedrank

The Omedrank algorithm by Fagin et al. [Fagin et al., 2003] is also an approach where multiple scalar quantizers are used to capture the proximity of descriptors in high-dimensional spaces. As it uses random projections, it runs into some of the problems discussed above with LSH. It differs from LSH because it does not uses hash tables and does not store descriptors into buckets. Instead, it uses B^+-trees to store the identifiers of the descriptors, not the descriptors themselves. The key difference, however, is that Omedrank *does not compute distances* to determine the near neighbors of query points. Instead, it uses rank aggregation.

III.3.1. Indexing and Searching with Omedrank

The Omedrank algorithm requires a pre-processing step in order to build several indices that are subsequently searched. First, a set of d' random lines is chosen (typically $d' \leq d$) and for each of these lines, a B^+-tree index is created and initialized. Each descriptor s is then projected onto each of the random lines. For random line j, a pair (id_s, v_s^j) is created, where id_s is the identifier of the descriptor and v_s^j is the value of the descriptor along the random line j. This pair is then inserted into the B^+-tree index for line j, which is ordered by the v_s^j values. When all the descriptors in the collection have been processed in this manner, the Omedrank index is ready and consists of d' B^+-trees, each containing n pairs of (id_s, v_s^j) values. It is key to note that the descriptors themselves (i.e., the values of their individual components) are not stored in the B^+-trees.

At query time, the query descriptor q is projected onto each of the d' random lines, giving a value of v_q^j for random line j. Each B^+-tree index is then probed with the appropriate value to find a starting point. Next, from this starting point, two cursors are started for each index, respectively reading successively lower and higher values. The cursors are used in a round-robin fashion, to simultaneously traverse all d' B^+-tree indices and retrieve the descriptor identifiers id_s. The algorithm keeps track of how often each descriptor identifier is encountered, while these cursors are moved. When a particular descriptor identifier has been seen in more than $\frac{1}{2}d'$ indices, it is returned as the nearest neighbor. Processing then continues, cursors are moved, until k descriptor identifiers have been returned.

III.3.2. The Case for B^+-trees and Median Rank Aggregation

Omedrank has few properties that makes it rather special in the panorama of high-dimensional indexing techniques. It has been designed by researchers from the database community and, therefore, special care is taken to nicely deal with I/Os and disks as Omedrank implicitly provides persistency of the indexed data on secondary storage. Data persistency is a by-product of using B^+-trees, which are one of the standard means to efficiently store and then access data that is kept on disks. B^+-trees are so central to databases that extremely efficient implementations exist. B^+-trees handle concurrent updates gracefully and are by essence balanced (in contrast to the buckets used in LSH that tend to be severely skewed).

While using B^+-trees is in fact a technicality, the inventors of Omedrank, however, took a very different design decision that makes it fundamentally different from many other indexing algorithms. Omedrank implements the *median rank* distance function to assess the similarity of high-dimensional descriptors, and [Fagin et al., 2003] shows it nicely approximates the Euclidean distance. Using median rank has several advantages that are:

1. *Efficient Use of the CPU to Determine Neighborhoods.* Median rank amounts to counting the number of times descriptor identifiers are observed. This is extremely cheap compared to calling a (Euclidean) distance calculation function for high-dimensional vectors. Omedrank therefore reduces as much as possible the CPU bottleneck that is typically observed with other techniques, including LSH.

2. *Cheap False Positive Removal.* Approximate k-nn search schemes might miss descriptors that are indeed near neighbors of the query points (hence producing false negatives) and might as well return descriptors that would not be part of the exact answer (hence producing false positives). Relying on the aggregation of ranks is an elegant and cheap way to eliminate some false positives, compared to standard geometry based approaches. Note that, however, aggregation of ranks is semantic-less, in contrast to using geometry. [Fagin et al., 2003] only returns descriptors that are seen along half of the random lines. This eliminates from the result distant descriptors having their projection close to the one of the query point, by accident, on one or on few lines. Note the impact of using other thresholds (e.g. $\frac{2}{3}d'$) has been studied in [Fagin et al., 2003] and [Lejsek et al., 2005, Lejsek et al., 2006a].

3. *Compactness.* Because Omedrank computes no distances, the high-dimensional vectors themselves do not need to be stored in the index. This saves considerable space. Since only descriptor identifiers are needed, few bytes are sufficient. This is particularly important when dealing with truly high-dimensional descriptors such as the GIST descriptors [Oliva and Torralba, 2001] or VLAD descriptors [Jégou et al., 2012] where many components must be accounted for. Having a compact index is also a desirable property since there exist as many copies of the data collection as there are projection lines. Only descriptors identifiers are in the copies Omedrank uses (in contrast to LSH which duplicates whole descriptors).

III.3.3. Problems with Omedrank

Omedrank is relying on scalar quantization. It therefore runs into the same problems as the ones observed for LSH. Random projections are not very well capturing the proximity of points in space, many projection lines are needed for quality. We compared in [Lejsek et al., 2005] the performance of Omedrank with the one of a well implemented sequential scan. The comparison gave insights on both the response time and the quality of the result.[2]

On the one hand, when very few projection lines are used, Omedrank is orders of magnitude faster than a sequential scan. On the other hand, Omedrank requires the use of a pretty large number of projection lines to return meaningful results. This is not surprising as this is consistent with the observations made with LSH. Omedrank is having, somehow, a behavior that is not as good as LSH. LSH has a notably more sophisticated way to create the final buckets into which descriptors are stored, compensating to some extent the problems of scalar quantization that often separates close descriptors and groups distant ones.

[2]The precise experimental set up allowing to draw those conclusions does not need to be repeated here—the phenomenon at stake are caused by the *scalar nature* of the quantization process.

III.4. Nearest-Vector Tree—NV-Tree

This section presents the NV-Tree. The NV-Tree is a high-dimensional index structure based on a scalar quantization process. It has specifically be designed from a database perspective: it can handle extremely large collections of descriptors and therefore it includes specific procedures to efficiently deal with secondary storage accesses, trying to alleviate the disk bottleneck. It also tries to mitigate the impact of the CPU bottleneck by avoiding costly distance calculations. The collections the NV-Tree indexes can also be concurrently updated while the system is running. It also deals with fault tolerance and includes specific mechanisms to enforce failure recovery.

The NV-Tree is one of the major contribution from the work we did during the last decade. This contribution gave birth to a start-up company, *Videntifier Technologies*. We successfully transferred the academic technology to *Videntifier Technologies* and hold two patents related to the core technology that is used. *Videntifier Technologies* now has 13 employees and is one of the main players in the forensics arena where tracking down illegal digital contents is paramount. Their search engine is currently able to index and identify videos from a collection of nearly 100 thousand hours of video, which is rather large for a system in production.

A brief recap of the history of *Videntifier Technologies* testifies of its good health. The company was founded in 2008, and was the winner of a business plan competition in Iceland. Soon after, the Icelandic police became the first client. In 2011, seed investment helped doing the first sales overseas. That same year, Eureka, the European organization running the Eurostars program gave support to a project entitled "Forensic Image Identifier and Analyzer" to add logo detection and identification to the *Videntifier Technologies* software suite, that technology being researched and then brought by INRIA. In 2012, *Videntifier Technologies* signed with Interpol, then with NCMEC (National Center for Missing and Exploited Children, USA) in 2013, series A financing. In 2014, *Videntifier Technologies* opened offices in San Francisco and Shanghai.

This section provides details on the design of the NV-Tree and reports various evaluations that were never published before, showing this high-dimensional indexing scheme has very nice performance.

III.4.1. Lessons from the Literature and DB-Oriented Motivations

The NV-Tree borrows from the literature and from the salient approaches that were state of the art a decade ago. It also borrows a lot from the motivations that are central to the database literature, where the management of data has to be in phase with practical use in the real world. There, acquiring data collection is costly and very time consuming, and loosing data because of an unexpected failure that can occur any time is not an option. There, also, updates to collections have to be supported. There, last, the I/O bottleneck has to be specifically tackled in addition to the usual CPU bottleneck to provide small response times and/or high throughput.

Designing the NV-Tree mainly took the factors that follow into account:

- We have seen with LSH and Omedrank that random lines were not successful at nicely capturing the proximity of the points in space. This is normal, random projection can only separate the points that are well laid out from the perspective of each random line. In practice, random projections tend to give the opposite; since collections are large and the dimensionality of data high, the likelihood of having distant points projected close from any perspective is extremely high, especially around zero. This is why both LSH and Omedrank need many lines to eventually obtain enough information to make possible this separation. As a side effect, maintaining many lines produces high redundancy, hence

severe storage requirements. Maintaining a lot of copies of the entire database is not a possible option—this is consuming too much space, especially with today's tera byte sized data collections.

- Traditional indexing schemes compute distances between points at query time, and, at large scale, this is a disaster. Worse, before that, descriptors must be fetched from disk, mostly randomly, killing performance. Then, once in memory, most of the time not wasted by I/O is wasted by distance calculations.
- Database literature says reading fixed size disk blocks is best for performance. It facilitates the prediction of the costs since the behavior of all reads is the same, it allows the operating system to better use prefetching and cache replacement policies. Overall, it reduces the variance of the response time of queries. LSH is not at all doing this. In contrast, the leafs of its index (the buckets) are all of different sizes, with an extremely high variance. This is detrimental to performance.

III.4.2. Design Decisions for the NV-Tree

The major six design decisions along which the NV-Tree is built are therefore:

Design Decision #1: Building on Random Projections

Although projections perform poorly in high-dimensional spaces, it has the immense advantage to turn mutidimensional features into single dimension values that can extremely efficiently be indexed by B^+-trees. In addition, we have decades of database experience optimizing B^+-trees as well as knowing how to safely update them concurrently.

Design Decision #2: Avoiding Having Many Copies of the Data Collection

When designing the NV-Tree, maintaining a lot of copies of the database could not be an option—this is consuming too much space. On the other hand, strongly reducing the number of projection lines (thus copies) almost automatically meant poor quality results. We wanted to use a large number of random projections (needed for quality) without paying the cost of the resulting data explosion.

We therefore designed a technique where projections were tightly coupled with segmentations: only the subset of the data collection that seems to be close in space from the perspective of one projection line is subsequently reprojected onto another line. This projection-segmentation strategy gradually cuts the space into regions where distant points are eliminated and where close points are kept for deeper analysis. It is like if LSH was projecting on independent new random lines the *contents of each of the buckets defined by a first hash function*, instead of *ignoring* what that first function produced when projecting the full collection on a second line.

Design Decision #3: Using Ranks to Circumvent the CPU Bottleneck and Rank Aggregation to Consolidate Results

Fetching descriptors from disk before using them in distance calculations is a very costly option—too much random I/Os and too much CPU. Ranking instead is much cheaper and the Omedrank approach proves it can simultaneously be a good approximation of the Euclidean distance and a nice and cheap way to eliminate some of the false positives resulting from the rough and imprecise scalar quantizations.

Design Decision #4: Using Fixed Size Disk Leafs to Circumvent the I/O Bottleneck

As some other indexing methods, the NV-Tree typically builds leafs containing data, leafs subsequently read to determine the result of queries. Following the database best practice, the leafs of the NV-Tree have to be of a fixed size. If possible, it is even better if they match the operating system I/O granule.

Design Decision #5: Supporting Concurrent Updates to the Index

Providing solutions allowing to insert new data points in high-dimensional indexes is not mainstream in the literature. It is usually not at all mentioned in papers, assuming users are happy with reconstructing the entire index from scratch when (enough) new data points are there. Schemes allowing inserts performed *concurrently* with searches are even more rare.

This is not acceptable from a database perspective. Database-oriented approaches naturally need to deal with disks and the persistency of the data they manage (this has been explained in the previous sections). They also need to deal with concurrent updates, this is discussed here. They also need to cope with failures, this is discussed next.

Another motivation for the need of concurrent updates is that some applications might not be able to stop running for index maintenance. We therefore believe that coping with updates while the system remains on-line is key for making the indexing scheme usable in practice, outside research labs.

Design Decision #6: Coping with Failures

While this seems completely obvious from a DB perspective, absolutely no paper about high-dimensional indexing ever discusses mechanisms to recover from failures. It is implicitly assumed the entire index can be recreated from scratch if a serious crash happens, ignoring disks catastrophic failures. This is not surprising, and goes well along with ignoring updates.

Not being able to cope with disks failures is not acceptable from a realistic point of view. It may take weeks to index a sufficiently large data collection, raw data may even not be available anymore, and making sure the index survives across crashes is key. We therefore believe special mechanisms to gracefully deal with failures must be incorporated in the indexing scheme.

We will describe how these design decisions materialize into the NV-Tree.

III.4.3. Principle of the NV-Tree

An early version of the NV-Tree, at that time called the PvS-index, was described in 2005, see [Lejsek et al., 2005]. The NV-Tree was first described in [Lejsek et al., 2006b], then deeply revised and enhanced in [Lejsek et al., 2009] and [Lejsek et al., 2011]. The NV-Tree is a disk-based data structure, which builds upon a combination of projections of data points to lines and partitioning of the projected space. By repeating the process of projecting and partitioning, data is eventually separated into small partitions which can easily be fetched from disk with a single disk read, and which are highly likely to contain all the close neighbors in the collection.

We first focus on index creation and retrieval. We then describe how the NV-Tree copes with typical database requirements such as concurrent updates and failure recovery. We then move to discussing new extremely large scale performance results that have never been published before—we show excellent recall values when managing about 30 billion high-dimensional descriptors.

Index Creation

Overall, an NV-Tree is a tree index consisting of: a) a hierarchy of small *inner nodes*, which are kept in memory during query processing and guide the descriptor search to the appropriate leaf node; and b) larger *leaf nodes*, which are stored on disk and contain references to actual descriptors.

When the tree construction starts, all descriptors are considered to be part of a single temporary partition. Descriptors belonging to the partition are first projected onto a single *projection line* through the high-dimensional space. See [Lejsek et al., 2009] for details about projection line selection strategies. The projected values are then partitioned into distinct sub-partitions based on their position on the projection line. Information about all the sub-partitions, such as the partition borders along the projection line, forms the root of the NV-Tree.

To build subsequent levels of the NV-Tree, this process of projecting and partitioning is repeated for all the new sub-partitions using a new projection line at each level, creating the hierarchy of inner nodes. The process stops when the number of descriptors in a sub-partition falls below a limit designed to be disk I/O friendly. A new projection line is then used to order the descriptor identifiers in each final sub-partition and the ordered identifiers are written to the leaf node, on disk.

Two partitioning strategies co-exist inside the NV-Tree and proved to work best. First, partitioning is such that the distance between partition boundaries at each level of the tree is equal. The normal distribution of high-dimensional vectors gives partitions with very different cardinalities and dense areas are partitioned deeper than sparse areas (same phenomenon as for LSH with the imbalance of buckets cardinalities). Second, the partitioning strategy changes when reaching the lowest levels of the tree: when a sub-partition fits into six leaf nodes, then data is partitioned one more time according to an equal cardinality criterion (instead of being based on distances). This results in better leaf node utilization and a shallower tree.

Nearest Neighbor Retrieval

During query processing, the search first traverses the hierarchy of inner nodes of the NV-Tree. At each level of the tree, the query descriptor is projected to the projection line associated with the current node. The search is then directed to the sub-partition with center-point closest to the projection of the query descriptor. This process of projection and choosing the right sub-partition is repeated until the search reaches a leaf node.

The leaf node is fetched into memory and the query descriptor is projected onto the projection line of the leaf node. The search then starts at the position of the query descriptor projection. The two descriptor identifiers on either side of the projected query descriptor are returned as the nearest neighbors, then the second two descriptor identifiers, etc. Thus, the $k/2$ descriptor identifiers found on either side of the query descriptor projection are alternated to form the ranked k approximate neighbors of the query descriptor.

III.4.4. Properties of NV-Trees

The NV-Tree indexing scheme has several key properties:

- *Single Disk Read:* Since leaf nodes have a fixed size, the NV-Tree guarantees query processing time of a single disk read regardless of the size of the descriptor collection. Larger collections need deeper NV-Trees but the intermediate nodes fit easily in memory and tree traversal cost is negligible.

- *Ranking and no Distance Calculations:* The NV-Tree returns approximate results in a ranked order. Returning a ranked result list has three major consequences. First, the descriptors themselves need not be stored within the leaf nodes, making it possible to store many descriptor identifiers in a single leaf node, which increases the likelihood of having actual neighbors in that leaf. Second, since no distance calculations are required, little CPU cost is incurred (scanning lists both directions), even for large collections. Third, as results are based on a projection to a single line, false positives do arise when processing a leaf node. Since distances cannot be calculated, other means of removing false positives are required.

- *Consolidated Result:* False positives can largely be eliminated by aggregating the results from multiple NV-Trees, which are built independently over the same collection. Since each NV-Tree is based on an independent pool of random lines, the contents of the ranked results are very likely to differ, except for the descriptors that are actual near neighbors. This idea of using multiple NV-Trees is somehow similar to what is done with LSH and Omedrank when several lines are used, each returning a partial result subsequently consolidated, with distance calculations for LSH and median rank aggregation for Omedrank.

- *Compact Data Structure:* The NV-Tree stores in its index the identifiers of the descriptors, not the descriptors themselves. This results in the NV-Tree being a very compact data structure. Compactness is desirable as it maximizes the changes to fit an entire tree in memory, avoiding I/Os and providing excellent performance.

- *Low Data Redundancy:* Both LSH and Omedrank require many lines to get high quality results, and thus they must maintain many copies of the database. With the NV-Tree, the repetitive process of projecting and partitioning within each tree eventually much better captures proximity in space than what LSH and Omedrank do. Hence, not so many NV-Trees are needed to get good results. Needing only few NV-Trees (typically 3 only) puts less pressure on storage as there are as much copies of the database as there are NV-Trees defined over the collection.

III.4.5. Concurrent Updates to the NV-Tree

This section presents a brief overview of the mechanisms used in the NV-Tree to allow the insertion of new data points into the index while searches run.

In order to know where a new data point has to be inserted in the index, then the very same procedure as the one used when searching for a query point is first ran. The NV-Tree structure is traversed, its internal nodes guiding the identification of the appropriate data leaf within which the data point should be inserted. That new data point is not inserted right away in the leaf stored on disk, but it is rather buffered in main memory and will be later and asynchronously pushed to disks. Buffering inserts obviously improves the performance as multiple inserts can be pushed at once to disks and according to an order that avoids complete randomness. Note that some logging is performed to enforce *Durability*, but this will be discussed later.

The NV-Tree is a multithreaded application and the thread pushing data to disks sleeps until the insert buffer gets filled enough. At that time, the insert thread determines from the contents of the insert buffer the most profitable leaf into which inserting data, and performs the insert(s). It also opportunistically inserts data in one data leaf when that leaf is loaded in memory because concurrent searches processing query points need that leaf. Overall, this tries to reduce as much as possible the I/O overhead for inserts.

Once the leaf into which a data point has to be inserted is in memory, then several conditions are checked. First, if that leaf is not entirely filled, then the identifier of the data point is simply inserted at the right location in the leaf, and the pages goes back to disk. When the data leaf is entirely filled, it must be split to provide more storage capacity within the tree. As it is generally recommended in the database literature, leafs storing data are created with some reserved empty space, their creation filling factor being roughly set somewhere between 50% and 85%. This leaves enough free space in data leafs to delay splits for some time. Furthermore, inserts tend to be evenly distributed across leaf pages, further delaying splits.

In order to efficiently deal with the case where the data leaf selected for insertion is full, then the insert process takes advantage of the hybrid partitioned structure of the NV-Tree: the projections lines at the bottom of the tree are partitioned according to an equal cardinality criterion while they are partitioned based on distances higher in the tree. Therefore, one data leaf at the bottom of the NV-Tree can naturally be grouped with its siblings into a subtree, rooted at the NV-Tree internal node where the partition strategy changes. There are of course many such subtrees in a complete NV-Tree.

When it is the first time any of the l pages of a subtree is full and can not accept an insert, then these l pages are reorganized in one go into $l+1$ pages. When a subsequent insert will find its target data leaf full, then the $l+1$ pages will be reorganized into $l+2$, and so on, until the number of pages in the subtree reaches a particular limit. At that time, the entire subtree is reorganized into (at least) two isolated groups, locally increasing the depth of the tree by one.

Reorganizing the contents of the data leafs from l to $l+1$ pages or when creating two or more distinct groups requires to reproject the descriptors onto random lines. When going from l to $l+1$ leaf pages, then redistributing the contents of those leafs nodes asks to first change the partitioning boundaries determined according to the even cardinality criterion and then to reproject all the descriptors. When splitting the overfull subtree into new groups, then a series of new random lines must be used to reproject the descriptors. Please note that the number of new groups depends on the distribution of the points onto the new projection lines.

Overall, it is thus mandatory at leaf reorganization time to use the values of the descriptors themselves and not only their identifiers which are kept inside the leafs of the NV-Tree. For efficiency, each subtree points to a special additional file where the original values of the descriptors are kept. When a subtree has to be reorganized, then its associated file is read and all the descriptors it contains are reprojected according to the new settings, boundary changes or new random lines. Each such file does not need to preserve any ordering of the descriptors it contains as the descriptors are always all processed at once. This facilitates the management of such files as they only need to be updated in an append-only way, which is both efficient and more robust to crashes.

III.4.6. ACID behavior for the NV-Tree

An enormous body of work exists in the database literature about concurrent inserts, coping with crashes and recovering from failures. From a conceptual point of view, the traditional notion of *transactions* enforcing the *ACID* properties[3] is required to make sure concurrent updates are properly handled as well as data not lost despite failures.

[3]It is out of the scope of this work to detail these four properties standing for *Atomicity*, *Consistency*, *Isolation* and *Durability* and that allow to clearly specify how a system should behave when performing concurrent updates while preserving an overall consistency and how it should behave to resist crashes. Note the *Consistency* property can safely remain undefined in this manuscript.

It is known that the complexity of the solutions enforcing the ACID properties largely depends on whether or not *multiple* inserts can be performed simultaneously and in concurrence with searches. Allowing multiple and simultaneous inserts triggers the implementation of extremely complicated procedures. But not all applications have such demanding requirements, and this greatly simplifies the implementation of inserts and significantly reduces the resulting overhead.

While we are convinced it is compulsory to enable concurrent updates, we also think that serializing the updates is for most CBIR applications enough, i.e., only one insertion transaction at a time should run concurrently with searches, that insertion transaction possibly inserting one after the other many new data points in the index. Said differently, no two insertion transactions should run concurrently, but it is fine if searches are performed while inserts are done. We therefore assume there are no multiple simultaneous inserts.

Because two inserts transactions can never conflict, then a simple B^+-tree-traversal based locking mechanism is sufficient to enforce the *Isolation* property. Both the insert and the search processes need to obtain read locks on nodes when traversing the tree, the locked parent being released once the child node is locked. Searching gets a read lock on the leaf page, released once its analysis is completed. Inserting must acquire a write lock on the whole leaf group to protect against concurrent reads and writes anywhere in that same group since splits are possible.

The *Atomicity* and *Durability* properties are enforced because we are using the seminal write-ahead logging protocol introduced by Gray [Gray et al., 1981] and then significantly extended by Mohan [Mohan et al., 1992]. Note the *Atomicity* property is only concerned with crashes because there is no need to worry about deadlocks, and inserts are not aborted. Serializing inserts also simplifies the enforcement of the *Durability* property.

Not surprisingly, several log files must be maintained, and are stored on separate hard disk drives. One log file keeps track of all the descriptors that are inserted. Since multiple NV-Trees can be used, each has to maintain its own index log file keeping track of the split operation it has to perform, to possibly undo and redo them. It should be obvious that splits are unlikely to occur at the same place in each NV-Tree since random lines are used.

Periodic checkpoints are taken to facilitate efficient recovery. At checkpointing time, the contents of the entire NV-Tree are flushed to disk (this includes the internal nodes of the tree, some possibly updated and/or split but not yet on disks) as well as the insert buffer.

During recovery, the latest checkpoint is used to recreate the NV-Tree in memory. Splits that were committed but not pushed to disk are replayed, others are ignored. All the descriptors inserted by uncommitted inserts that reached disks are removed and committed inserted descriptors that did not reached the disks are re-inserted.

Each NV-Tree logs and checkpoints itself independently from the other NV-Trees, avoiding complicated synchronisations issues. They must however run insert transactions according to the same order (but not at all at the same time) to make sure recovery recreates a globally consistent state. An LSN type-of mechanism is fine for that task.

III.4.7. Performance of the NV-Tree

Single Descriptor Experiments

This section provides details on the performance of the NV-Tree when searching the nearest neighbors of single query points. Only a subset of these performance measurements have already been published elsewhere, see [Lejsek et al., 2009, Lejsek et al., 2011].

The results at very large scale are new.

To evaluate the query performance of the NV-Tree, we use the ground truth that has been

Set Name	# of SIFT Descriptors	Size on disks	# of images	Size of NV-Tree
30M	28,799,690	3.8G	26,583	180M
180M	179,443,881	23.6G	179,277	1G
300M	305,443,749	40.3G	334,268	1.9G
2.5G	2,485,568,191	328G	2,259,277	14G
3G	3,040,856,472	401G	2,970,596	17G
30G	28,484,904,924	3.7T	28,969,271	162G

Table III.1.: Figures for the image collections used in this manuscript. Resolution= 512 pixels

defined earlier in this manuscript, Section II.3.4, page 20. We recall here that this ground truth identifies 248,212 descriptors being the contrasted enough nearest neighbors of 500k query points. The precise description of the evaluation protocol used here is given Section II.3.5, page 21.

We are drowning these 248,212 descriptors into descriptors sets having different cardinalities. These sets of somehow *distracting* descriptors have been created by extracting low level features from up to 30 million images randomly downloaded from Flickr. The experiments in this section are about recall, i.e., how many of these 248,212 ground truth descriptors are found when using the set of queries, varying the number of distractors. We used distractor sets containing about 30 million descriptors, 180 million, 300 million, 2.5 billion, 3 billion and 30 billion. This has been presented above, Section II.4.3, page 23.

We are not aware of any other experiment ever published where recall measurements are obtained when searching the nearest neighbors of individual query points lost within up-to about 30 billion distracting descriptors. For the sake of precision and for the ease of reading, Table III.1 repeats the exact figures for this experiment. From this table, it is clear that the design of LSH has flaws precluding it to work at large scale. It can not cope with managing (reading, updating) hundreds of copies of a 3.7TB dataset containing 30 billion SIFT descriptors stored in a (compact) binary format...

Figure III.2 shows the recall for various collections varying in their sizes. It also shows the impact on recall when coupling a varying number of NV-Trees. Up to 3 NV-Trees were used against all the datasets; as such experiments are complicated and time consuming, we used only two moderate size datasets for checking the impact on recall of using up to 6 NV-Trees.

When using a single NV-Tree, recall is pretty low. Close to 54% of the 248,212 ground truth descriptors are found when they are lost in a collection made of about 30 million distracting descriptors. This percentage slowly decreases as the distracting collections grow, and it ends up at about 38% when challenged by the 30 billion distracting descriptors collection.

Using additional indices dramatically improves performance. With the 30M collection, recall jumps to 72% with two NV-Trees and reaches 79% when using three NV-Trees. At the other end of the figure, with 30G, recall diminishes but remains remarkably good given the size of the distracting collection: it is 52% when two NV-Trees are used and 58% with three NV-Trees.

Using more than three NV-Trees provides a slight recall improvement, but not as dramatic as the ones we observed when going from a single to two and three NV-Trees. Multiplying the number of NV-Trees is not a worthy option: it increases the pressure on the storage as multiple copies of the indexed collection must be maintained (see the last column of Table III.1; note the descriptor collection itself never has to be duplicated); it makes the retrieval cost higher as multiple trees must be probed.

Overall, from a quality perspective, using three NV-Trees is the best option. It proved to work extremely well even when dealing with a collection that is truly large scale and the experimental

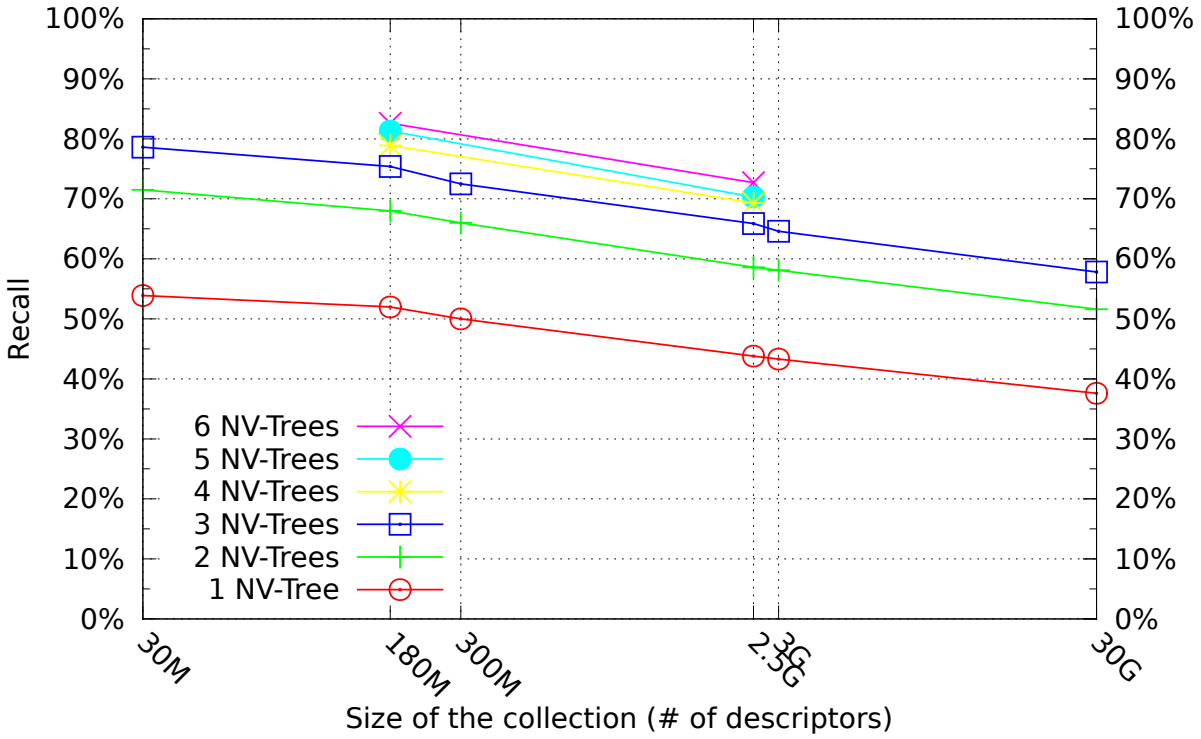


Figure III.2.: Recall, varying dataset sizes, single descriptor experiment

set up extremely challenging: the NV-Tree can correctly identify most of the nearest vectors of a unique query point even when these vectors are lost in the middle of 30 billion distracting descriptors. With three trees, processing cost is moderate, the recall is excellent, the storage demands are reasonable. A little less than 500GB are needed to keep three NV-Tree indexing the 30 billion data collection.

We now turn to the retrieval performance. The numbers we are going to talk about have been acquired while running the experiments described above. We measured the response time of each individual query as well as the throughput of the system, determining the number of query descriptors it can process per second. These numbers follow very closely what was observed in [Lejsek et al., 2011], slightly differing however because the hardware is different. Trends, however, are identical. We therefore do not go into the details but simply present the key retrieval performance elements, where the dominating costs are related to the CPU consumption and main memory latency when the NV-Tree indices fit in main memory, and to the performance of disk reads when indices do not fit.

We ran experiments using a Dell r710 machine that has two Intel X5650 2.67Ghz CPUs. Each CPU has 12MB of L3 cache that is shared by the 6 actual and 6 virtual cores. There are therefore 24 cores, 12 being real processing units. The RAM consists of 18x8GB 800Mhz RDIMMs chips for a total of 144GB. That machine is connected to a NAS 3070 from NetApp offering about 100TB of disk space, raid-ed. We ran the experiments using a single core. Some experimental settings involve multiple NV-Tree; they are probed one after the other—no parallelism is enforced here while this could be trivially done. We focus on using up to three NV-Trees at most. When the various indices entirely fit in main memory (this was the case for all experiments but the 30G

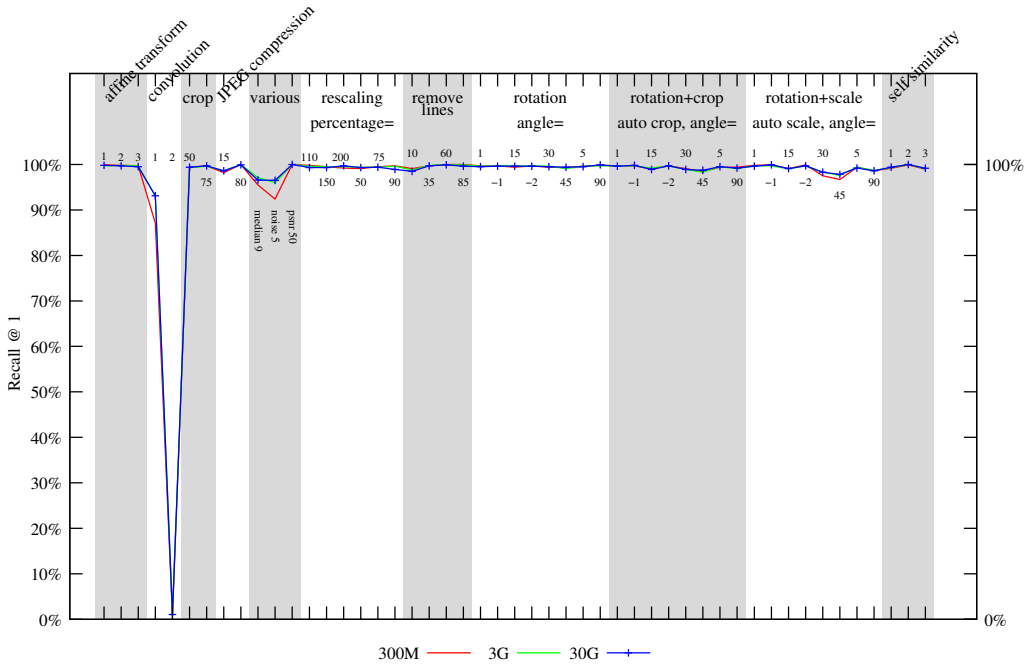


Figure III.3.: % of images found, 49k query set, varying distracting datasets

one), then answering each query descriptor is extremely fast. It takes a fraction of a millisecond to process one descriptor against one NV-Tree, and the throughput we observed ranged between 2,000 and 3,000 query descriptors per second per tree. Note such good performance could be achieved only once each NV-Tree index entirely resides in main memory. Filling the memory can be forced before starting running queries, or it can be a consequence of the querying process. In this later case, the first queries to run are slow as they need to fetch data from disks, subsequent queries are faster as they more and more likely find the data they need in the cache, loaded by earlier queries. The memory of the machine we used is big enough to eventually store up to three NV-Trees associated to the 3G experiment.

No index can entirely fit in main memory when using the 30G collection. In this case, the system has to get data from disks for almost every query descriptor. Each is likely to access a different part of the index, randomly, and no main memory buffering policy copes with such demanding access patterns. The response times are thus much bigger. The duration of each I/O varies but stays in the order of 5 to 20 milliseconds. I/Os are very random and it is extremely complicated to precisely know the way the NAS NetApp server handles them. It serves many users in parallel, has various level of caches that we can hardly control and observe, and stripes the data across its disks out of our sight. Overall, however, about 50 query descriptors could be processed per second per tree.

Image-Level Experiments

This section provides details on the performance of the NV-Tree when searching for similar images. In this case, the queries are made of numerous descriptors all together used to find images that are similar. Two series of experiments were made. The first one corresponds to the experimental protocol named “49k” as defined in the Section II.4. The second series of experiments corresponds to the experimental protocol named “Copydays”.

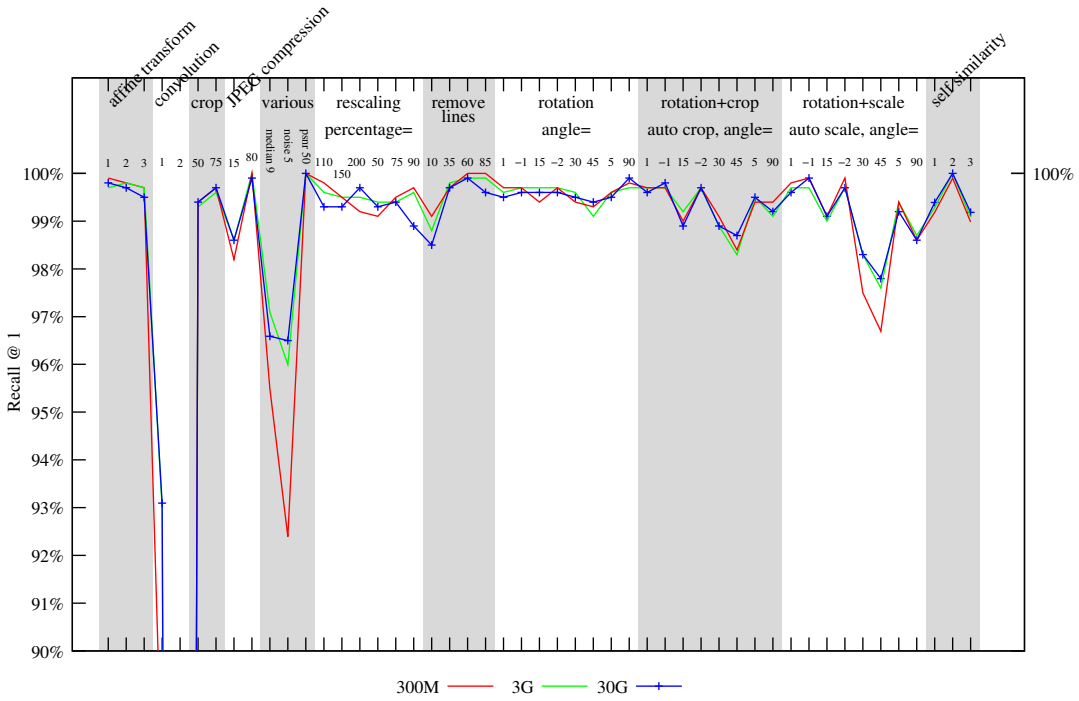


Figure III.4.: % of images found, 49k query set, varying distracting datasets. Zoom on 90–100%

49k Image Benchmark. We first discuss "49k"-related experiments. The datasets and the ground truth we have been using in this experiment are the ones detailed Section II.4. Figure III.3 shows the percentage of ground truth images that are found when querying 3 NV-Trees with queries resulting from the 49 transformations grouped by categories (defined Section II.4) along the x-axis. As we said at ground truth definition time, the search process is declared *successful* if the correct ground truth image has rank #1 in the result list when using the appropriate query. When the image at rank #1 *is not* the original image, then the system is said to *fail*, even if that image turns to be at rank #2. For one type of transformation, 100% means that all the 1,000 ground truth images were found at rank #1 in the result set.

It can be seen from this Figure III.3 that image retrieval works extremely well, and that almost all ground truth images are found when queried with images belonging to the 49k image benchmark. This observation applies to the three cases where the ground truth is drown into distracting collections of varying sizes, ranging from 300 million descriptors (i.e. 334,268 distracting images) to 30 billion descriptors (i.e. 28,969,271 distracting images). The lines giving the performance of the system with the 30 billion descriptors distracting set is plotted in front as this is the most challenging set up. Figure III.4 is zooming on the results lying in the range of 90% to 100%. Few comments are in order, however.

First, one image variant is almost never found, no matter the distracting image collection. It is the variant referred to as "CONV_2" in Section II.4. This image modification produces almost completely black images where close to no detail remain. It is frequent that the computation of the SIFT descriptors on these images produces no SIFT at all, or only one, two at most. When the queries are made with so few descriptors then only luck can bring the ground truth image at the top of the result list (it is sometimes elsewhere in this list, sometimes not present at all).

Second, 100% is not always reached for some variants (for the 30G case, "JPEG_15" gives 98.5%, "RML_10" gives 98.4% and 96.5% with "MEDIAN_9"). A detailed analysis of the result

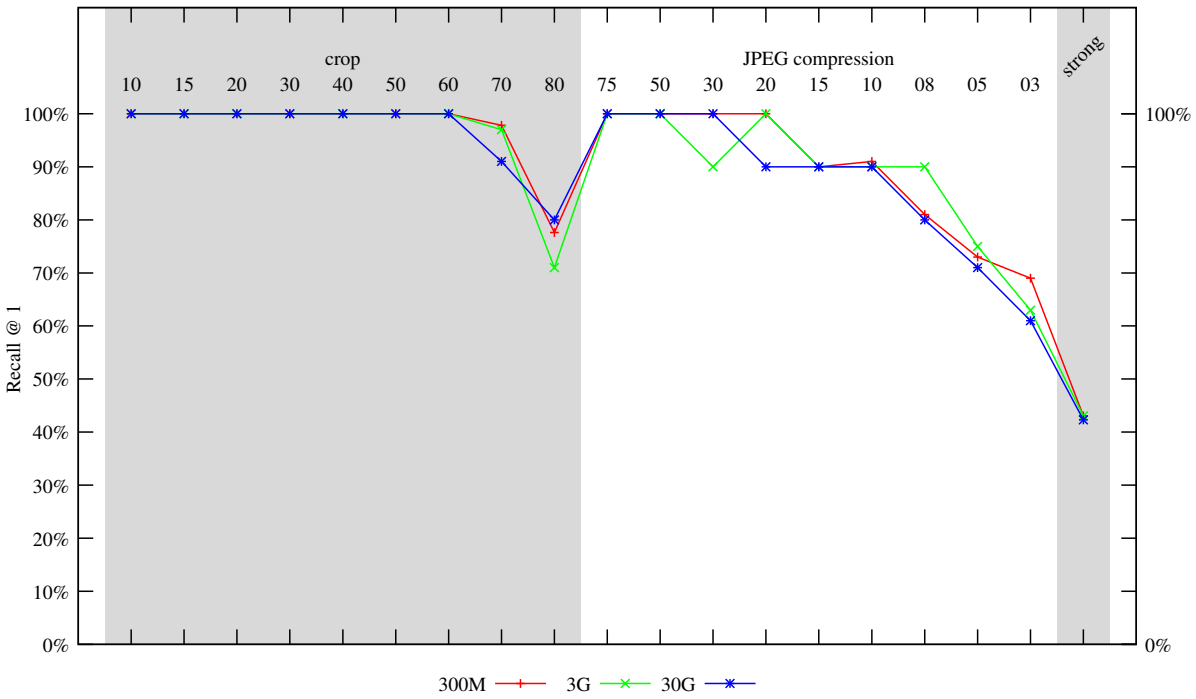


Figure III.5.: % of images found, Copydays query set, varying distracting datasets

lists show that in *all such cases*, then the ground truth images have ranks #2 (this is the most frequent situation) to #5. This is an overall extremely good result, especially because our success criterion is very strict as it considers only rank #1.

Third, the experiments using the 30G set do not always return the worse results. A detailed analysis shows that results tend to be bad when the indexed collection is very small. In this case, there are too few projections and segmentations to nicely separate unrelated data points while keeping together close ones. This obviously does not happen with larger collections. These larger collections are in contrast quite challenging because many distractor data points might produce noisy information along projection lines. It is interesting to note the performance of the NV-Tree improves when the collections to index get larger and larger.

Copydays Image Benchmark. Figure III.5 gives similar results but for the Copydays benchmark image dataset (see II.4). This benchmark contains image transforms that are more difficult to identify. Here again, the y-axis gives the recall at 1. It can be seen that the results are excellent... The NV-Tree is able to identify most of the time the correct images from even quite strongly distorted queries. It is not surprising to observe that quality drops with very severe crops, with extremely compressed images (a human being can sometimes hardly find any similarity between a JPEG 3% compressed image and its original counterpart) and with some of the strong variants. Note that sometimes such attacked query images create a handful of descriptors, making the identification of the original counterpart extremely challenging. Sometimes, there are too few matches to clearly have the original at rank #1—it is lost in the noise.

A Note on some Optimizations

We added to the NV-Tree various optimizations that proved to reduce the costs of processing queries. We think one deserves attention. This optimization applies to the case where one index is used to run image-level queries, i.e., when it is known that *a set* of descriptors must be used to find similar images. This is not applicable to the case of single descriptor settings.

When concerned with image recognition based on local descriptors, query time is often high because hundreds of descriptors are used to probe the database at search time. In [Lejsek et al., 2006a], special stop-rules have been designed to abort as soon as possible the search process, trying to use only a small subset of the descriptors in the query. In a copyright enforcement scenario, stop-rules assume that pirated images receive a lot of descriptors matches, while unrelated images receive few random matches, roughly distributed over the whole image collection. Therefore, after having processed a fixed number of query descriptors, a probability-based stop-rule aborts the processing if one particular image very rapidly collects many matches; another rule terminates the search if all scores stay roughly equal and low, suggesting no copy detection. In [Lejsek et al., 2006a], as few as 20 descriptors matches (this is roughly 2%) were found sufficient for finding a copy, and about 100 descriptors had to be used to decide it is not a copy.

A Note on Comparing with LSH

Almost none of these experiments could have been ran with the standard LSH algorithm. LSH must store in the buckets all the components of all the descriptors for the final distance calculations, and many copies of them must be maintained, one per random projection line. When 100 hash tables are needed, then even the smallest collection corresponding to the 30M settings can not fit in memory. It is needless to detail the storage requirements that would be required with LSH when indexing the 30G dataset, each copy occupying 3.7TBytes.

[Lejsek et al., 2009] gives elements of comparison, with different settings, however. Overall, it was observed that LSH needs many more projection lines than you need NV-Trees to get a recall that is as good. This is normal: each projection line of LSH can not do a good job at representing the proximity of points in space as the entire collection is projected at once. It is only the combination of many lines that works. The NV-Tree combines many lines in a single index since the data collection is repeatedly projected and partitioned.

LSH eliminates false positives via distance calculations. The NV-Tree does this by consolidating ranks. This second option is faster, but for sure less accurate.

III.5. Concluding Chapter III

From our point of view, the NV-Tree is a significant contribution to the field of very large scale high-dimensional indexing. It can index collections of extremely large size as it includes mechanisms to gracefully cope with secondary storage. In addition, the foundations of that system are such that it can be used in the real world: it resists failures thanks to the implementation of an ARIES-based protocol, it supports dynamic updates, it has a balanced behavior facilitating resource planning.

The NV-Tree left our research labs a while ago, and has been turned by *Videntifier Technologies* people into an operational and commercial system enforcing the tracking of illegal contents on the Web. The NV-Tree was extended to cope with videos and several specific algorithms for efficiently processing keyframes have been implemented. In addition, it now includes schemes for identifying logos overlaid on the video, facilitating and speeding-up the identification of contents.

To meet performance requirements, the NV-Tree includes specific image processing techniques implemented on a GPU processing board. The NV-Tree is a success.

A lot of time has passed since we took the first design decisions for the NV-Tree (called PvS at that time, see [Lejsek et al., 2005]). Its design closely matches our database perspective detailed Section I.4; in particular, we designed the NV-Tree such that it is a very generic high-dimensional searching tool, returning the approximate k-nn of single descriptors.

It turns out it is used every day at *Videntifier Technologies* to run searches at the level of images. In this case, other indexing mechanisms can be used, such as the ones presented in the next chapters where *aggregating* descriptors offers a lot of space savings for impressive performance improvements.

Since the first design decisions, the database as well as the computer vision communities have acquired a lot of understanding on high-dimensional indexing. New high-dimensional indexing approaches have emerged since, and it is the purpose of the next chapter to review them and to discuss our contributions.

Chapter IV

Vectorial Quantization Based Indexing Techniques

Contents

The previous chapter detailed several indexing solutions built around the notion of scalar quantization. They work, both in theory and in practice, and at least one scheme, LSH, has become one of the most popular indexing solution used in many projects, worldwide. It is however clear that all these approaches require specific mechanisms to compensate for their poor ability to nicely capture the locality of the data in the feature space. These mechanisms have a cost in terms of computational and/or space complexity, and this cost is often non negligible. For this reason, researchers have investigated the use of vectorial quantization schemes instead of scalar ones. Vectorial quantization techniques better capture the co-locality of high-dimensional feature vectors in space. The quality of indexing solutions built on top of vectorial quantization foundations is therefore improved. Their nice properties are such that most of the state-of-the-art indexing solutions now use vectorial quantization processes.

This chapter describes our experience in working with vectorial quantization based indexing

techniques. We have implemented and evaluated several members of the vectorial quantizers family, and we report here their performance, both compared with scalar quantizers as well as comparing vectorial techniques one against the others.

Historically, this work comes after the work we did on scalar quantization techniques. This work has been done with people originating from domains that are quite far from databases, such as the domain of compression and signal processing, enriching the work with their points of view and borrowing techniques established in their respective domains but never quite applied to high-dimensional indexing. Overall, it can be observed that our database perspective has a shallower penetration on the techniques we designed all together. Nevertheless, caring for I/Os, facilitating resource consumption planning, optimizing for response times or throughput are all issues taken into account at various degrees in the schemes we will detail in this chapter. What is the most missing is the enforcement of durability. One reason explaining this lack of support for durability is that we had no technology transfer goal in mind while discovering the world of vectorial quantization based high-dimensional indexing techniques.

This chapter is structured as follows. We first introduce the general principles of vectorial quantization based indexing techniques in Section IV.1. Section IV.2 briefly presents the principles followed by *structured* vectorial quantization schemes that partition the feature space according to some kind of fixed geometrical structure, such as a lattice. We then move in Section IV.3 to present *unstructured* vectorial quantization schemes that to some extend take into account the actual distribution of the data in space for partitioning; this section also compares the performance of several schemes. We draw lessons in Section IV.4 and propose several extensions that address some of the problems observed with vectorial quantization schemes when taking a database perspective. Last, Section IV.5 concludes this chapter.

IV.1. Introduction

In contrast to scalar quantization where the code-book $\mathcal{F}$ is defined with scalar mono-dimensional values, vectorial quantifiers for high-dimensional indexing typically define $\mathcal{F}$ in a high-dimensional space. The representatives values $\hat{x}_i$ are therefore not scalar values, but high-dimensional vectors. They typically lie in $\mathbb{R}^{d' \geq d}$ when $x \in \mathbb{R}^d$.

The representative values are typically used to create a Voronoi diagram. In general, the original feature space is partitioned into cells, each cell being associated to a representative value produced by the quantizer. Furthermore, one cell contains of all the data points that are closer to the cell's representative than to any other representative. The sizes (in space) of the cells will depend on whether or not representatives are regularly located in the feature space. When representatives are regularly laid out, then the associated vectorial quantization schemes define regular lattices. In this case, cells have typically all the same size/volume in space. We describe few examples of such lattices next. When no particular regularity is imposed to representative values, then algorithms such as k-means can be used to materialize the quantization. In this case, cells may have different volumes. We describe such approaches below.

The typical usage of vectorial quantization schemes for indexing is as follows, using again a very simplified view, however. A set of representatives is determined and used to partition the entire data collection. Each point of the data collection gets assigned to the representative that it is the closest to. This eventually partitions all the descriptors into groups. At search time, the representative that is the closest to the query point is identified. Then, all the descriptors that have been assigned to that representative are scrutinized to possibly be part of the final result that will be returned to the user.

IV.2. Structured Quantization

We now move to describing structured vectorial quantization schemes, also known as high-dimensional lattices.[1] We first briefly review some lattices that have been used in the context of high-dimensional indexing. We then detail some of the problems such approaches have.

IV.2.1. Lattices for High-Dimensional Indexing

Lattices have been extensively studied in mathematics and physics. They were also shown to be of high interest in quantization [Gray and Neuhoff, 1998, Conway and Sloane, 1982b]. For a uniform distribution, they give better performance than scalar quantizers [Gray and Neuhoff, 1998]. Moreover, for some lattices, finding the nearest lattice point of a vector can be performed with an algebraic method [Agrell et al., 2002]. This is referred to as *decoding*, due to its application in compression. Lattices have been used for indexing due to the proximity of the problems they address.

A lattice is a discrete subset of $\mathbb{R}^{d'}$ defined by a set of vectors of the form

$$\{x = u_1 a_1 + \cdots + u_d a_d | u_1, \cdots, u_d \in \mathbb{Z}\}$$

where $a_1, \cdots, a_d$ are linearly independent vectors of $\mathbb{R}^{d'}$, $d' \geq d$. Hence, denoting by $A = [a_1 \cdots a_d]$ the matrix whose columns are the vectors a_j, the lattice is the set of vectors spanned by Au when $u \in \mathbb{Z}^d$. With this notation, a point of a lattice is uniquely identified by the integer vector u.

Lattices offer a regular infinite structure. The Voronoi region around each lattice points has identical shape and volume (denoted by $\mathcal{V}$) and is called the *fundamental region*.

When used in the context of indexing, it is key to quickly find the appropriate lattice point given one input vector. This input vector is either one of the vectors of the data collection to index, and in this case finding the lattice points allows to determine in which cells it has to go. Or the input vector is a query vector and finding the lattice point allows to know which cell to analyze and thus in turn which vectors from the data collection to scrutinize. As decoding speed is key, we focus here on some particular lattices for which fast decoding algorithms are known. These algorithms take advantage of the simplicity of the lattice definition. We briefly introduce the lattices D_d, D_d^+ and A_d. For these lattices, finding the nearest lattice point of a given query vector is done in a number of steps that is linear with its dimension [Conway and Sloane, 1982a]. More details can be found in [Conway and Sloane, 1987, chap. 4].

- *D_d:* The lattice D_d is the subset of vectors of $\mathbb{Z}^d$ having an even sum of the components:

$$D_d = \{(x_1, \cdots, x_d) \in \mathbb{Z}^d : \sum_{i=1}^{d} x_i \text{ even } \}, d \geq 3$$

- *D_d^+:* The lattice D_d^+ is the union of the lattice D_d with the lattice D_d translated by adding $\frac{1}{2}$ to each coordinate of lattice points. That translation is denoted by $\mathbf{\frac{1}{2}} + D_d$.

$$D_d^+ = D_d \cup (\frac{\mathbf{1}}{\mathbf{2}} + D_d)$$

[1] There is another family of structured quantizers dividing high-dimensional spaces into hypercubes. These are members of the Space Filling Curves family. [Moon et al., 2001] proposes a study analyzing the properties of the Hilbert space filling curve known for its good performance. Such curves, however, are out of the scope of this manuscript.

When $d = 8$, this lattice is also known as E_8 Leech lattice [Shakhnarovich et al., 2006], which offers the best quantization performance for uniform 8-dimensional vectors.

- *A_d:* The lattice A_d is the subset of vectors of $\mathbb{Z}^{d+1}$ living on the d-dimensional hyper-plane where the sum of the components is null:

$$A_d = \{(x_0, x_1, \cdots, x_d) \in \mathbb{Z}^{d+1} : \sum_{i=0}^{d} x_i = 0\}$$

A vector q belonging to $\mathbb{R}^d$ can be mapped to its $d+1$-dimensional coordinates by multiplying it on the right by the n lines $\times$ $n+1$ columns matrix:

$$\begin{pmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ . & . & . & \cdots & . & . \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{pmatrix}$$

Lattices have nice spatial consistency properties making them interesting for indexing. We refer to these properties as the *vectorial gain*, which is a combination of the two following observations:

1. With lattices, any two points decoded to the same lattice point are separated by a bounded distance, which depends only on the lattice definition. This is not at all the case for scalar quantization schemes as the maximum distance between points in the feature space is not bounded, even when they are associated to very close (or identical) projected values. This bound on distances with lattices thus much better capture the co-locality of points in space.

2. One lattice might have a stronger vectorial gain than another lattice as vectorial gain is strongly related to the *density* of lattices. The density of a lattice is the ratio between the volume $\mathcal{V}$ of the fundamental region and the volume of its inscribed sphere. Basically, considering Euclidean lattices, the closer to 1 the density, the closer to a sphere the fundamental region, and the greater the vectorial gain. Figure IV.1 illustrates the vectorial gains for the $\mathbb{Z}^2$ lattice (a) and the A_2 lattice (b) having the volume of their fundamental region being identical. Max_a and Max_b are respectively the largest possible Euclidean distance between any point of the lattice and the center of the cell. Here, for identical volumes, $\text{Max}_a > \text{Max}_b$ because a $\mathbb{Z}^2$ lattice is less compact than a A_2 lattice.

There are essentially 4 parameters to set when using one specific lattice for high-dimensional indexing. The first parameter is obviously the *nature* of the lattice, which in turn defines the shape of its fundamental region. Then, a *scale* parameter w is to pick, and this impacts the size of the fundamental region. It is also possible to define a *rotation* parameter r that defines the orientation of the lattice with respect to the natural axes along which the data collection is described. Finally, a *translation* factor might be applied to shift the data with respect to the origin of the lattice. Using a $\mathbb{Z}^2$ lattice, Figure IV.2 illustrates[2] the effects of a translation (a), a rotation (b) and a change of scale (c). It is assumed here that having most data points in one single cell gives better results as no near vector stored in any neighboring cell gets ignored. On this illustration, the translated lattice can only give worse results as the cloud of point is split. In contrast, the rotated lattice better captures the cloud of points. Finally, the scale change scatters the data points into many cells.

[2]This illustration is just a toy example.

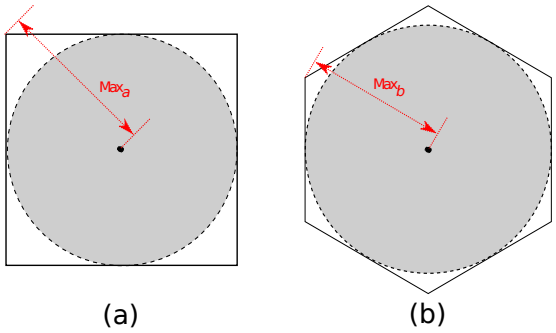


Figure IV.1.: Illustrating the vectorial gain with fundamental regions of identical volume, lattice $\mathbb{Z}^2$ (a) and lattice A_2 (b). Inscribed spheres are in gray

Lattices in Action

Mejdoub et al. proposed in [Mejdoub et al., 2009] an high-dimensional indexing scheme built from $\mathbb{Z}^n$ lattices. They propose to use a tree of lattices where a large scale lattice at one level embeds another finer scale lattice at a the level below. This quantizes the data vectors progressively into smaller partitions using a finer scaling factor. The proposed approach is efficient when running similarity queries because it uses the hierarchy and the good algebraic and geometric properties of the lattices. Overall, despite ignoring many key details of their approach, the most elaborated scheme they propose works as follows.

All features vectors are first quantized into a $\mathbb{Z}^n$ lattice using a rough scaling factor γ_1. The lattice vector x quantizing the data collection vector v is determined by computing:

$$x = \left[\frac{v}{\gamma_1}\right]$$

where $\left[.\right]$ stands for the "round" operator; γ_1 is the first scaling factor used (set to $1/2$ as described in [Mejdoub et al., 2009, Section 6].

Then, for each lattice cell that is non-empty, additional rounds of $\mathbb{Z}^n$ quantizations are performed. All vectors falling in one cell at level i using a γ_i-scaled lattice are re-quantized, forming the level $i+1$ using a $\gamma_{i+1} = \gamma_i/\alpha$-scaled lattice, α being an odd value (typically 3). A subset of the dimensions of vectors are used when moving from level i to level $i+1$, the ones that least spread the points to re-quantize. This process goes on until the lattice tree has reached a predefined maximum number of levels (3, see [Mejdoub et al., 2009, Section 6]).

At query time, the query vector goes through the tree of embedded lattices. At each level, a bit-permutation based technique allows to quickly find the best neighboring cells that are worth visiting in addition to the ones directly determined by the quantization steps.

The performance experiments reported in [Mejdoub et al., 2009] show this scheme efficiently copes with collections containing one million local descriptors. This is not a lot compared to the collections we are used to manipulate. It is unclear whether this lattice embedded technique still works at a larger scale. The main reasons are linked to the intrinsic properties of lattices that eventually fail; this is discussed below in Section IV.2.2.

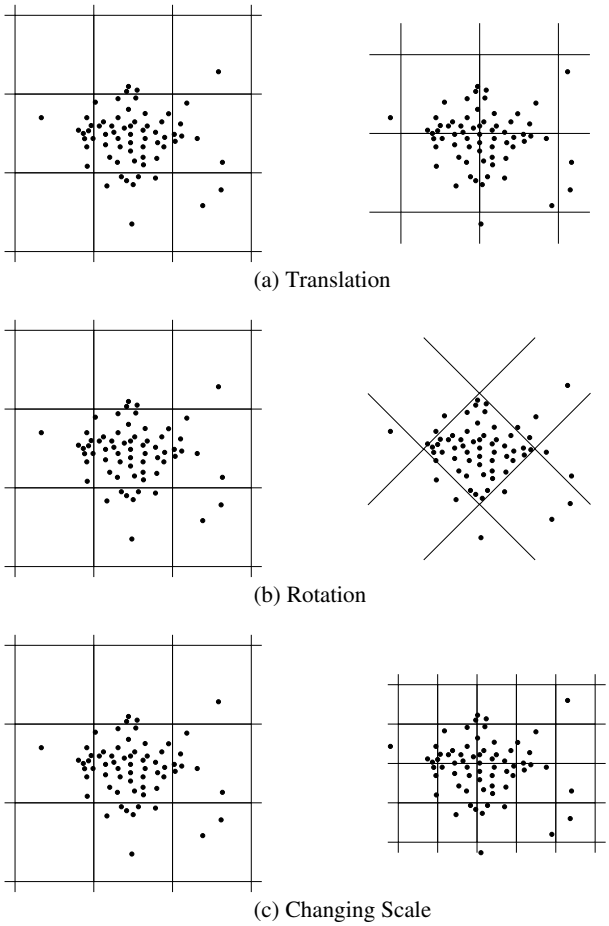


Figure IV.2.: Effects of (a) translating, (b) rotating and (c) changing the scale of a lattice $\mathbb{Z}^2$

IV.2.2. Problems with Structured Quantizers

Non-Uniform Data: Empty Cells, Full Cells

Lattices (and Space-Filling Curves) create a perfect regular partitioning of the data space. All fundamental regions are Voronoi cells that all have the exact same size in space and volume. That regularity enables the existence of fast decoding procedures. On the down side, that regularity does not nicely cope with the distribution of any point collection constructed from real world data. Real data is typically non uniformly distributed, and some parts of the feature space tend to be particularly dense while others are sparse. In turn, all structured quantizers used to index real data sets exhibit a severe variance in the cardinality of the cells containing points. Some cells contain very few data points while few others contain an extremely large number of points.

This phenomenon was already observed when we discussed the problems into which scalar quantization schemes run—same causes, same effects. In [Paulevé et al., 2010], we replaced the hash functions scalar quantizers of LSH by lattice-based vectorial quantizers. We checked (among other things) the cell population for A_{128}, D_{128} and D_{128}^{+} using a data collection comprising one million SIFT vectors extracted from real images. Few cells contain more than 10,000 points each, about 100 cells contain above 1,000 points each while about 100,000 cells contain less than 10 points.

Complicated Parameter Tuning

Determining which lattice to use and then at which scale, rotation and translation is an extremely complicated process. Very good values for those parameters can probably be found for best indexing some part of the feature space. It is unlikely those same values will do a good job globally, as the feature space typically contains regions that are extremely different one from another, in terms of density for example.

In [Paulevé, 2008], Loïc Paulevé ran a quite large number of experiments where the scale factor, the rotation angle and the translation used together with various lattices were varied. He observed the variance of the cardinalities in fundamental regions slightly changes depending on those lattice parameter values, but there was simply no way to find any parameter combination balancing the cells. Loïc Paulevé also observed that reducing the dimension of the data helps but does not solve the problem.

Overall, it is very hard to tune lattice parameters to simultaneously get few empty cells and no cell with too many points. Balancing the cardinality of cells can probably not be achieved by tuning those parameters, it is hopeless.

Lattices for Specific Dimensions Only

Some lattices and their fast decoding procedures do not exist for any arbitrary dimension. For example, it has been demonstrated that the best possible lattice for $d = 24$ is the Leech lattice [Shakhnarovich et al., 2006], as it is when $d = 8$. But no fast decoding procedures exist for other dimensions. It can therefore not be used to index any data collection, or it is required to reduce the dimension of the descriptors to match the optimal definition of the lattice, which might not be desirable.

Impracticable Enumeration of Neighbors

At search time, the query point is used to determine which lattice cell is to analyze. When the determined cell turns out to contain very few points, and this is likely to happen frequently, then the result quality can hardly be good. It makes sense to analyze the points that are stored into the neighboring cells as they may contain quite good neighbors. Reading more than one cell is an idea that can be found everywhere in indexing strategies: we discussed it in Section III.2.3. It has also been applied to vectorial quantization schemes as proposed by [Philbin et al., 2008].

This simple idea turns out to be very hard to instantiate in practice when using lattices. Which additional cell(s) should the search process analyze? There are many cells around the one determined from the query point, and this number is linked to the geometrical properties of the lattice used. It is an dimension-exponentially complicated task to enumerate all the neighboring cells. Furthermore, all cells are at the same distance and there it is therefore hard to determine which one is the first to start with in order to best increase the quality result. Note that some of those cells might contain very few points as well, or might even be empty.

In his work, Loïc Paulevé implemented several strategies for increasing the number of neighboring points to analyze without enumerating all the neighboring cells [Paulevé, 2008]. He used some kind of lattice embedding scheme where large grain lattice cells facilitate determining which finer grain lattice cells should be processed. He also used schemes where he created multiple lattice based indexes differing by their scale, orientation and translation, but all indexing the same data collection. At search time, he used several strategies to pick for example the cell where the query point is the closest to the center, or to check the union of all cells belonging to different indices where the query point falls. Overall, result quality is improved with such

strategies, at the cost of multiple decodings and accesses to cells. Another strategy was implemented, more sophisticated, as it was ordering the neighboring cells to analyze depending on the distance to their facets. First analyzing the cell associated to the nearest facet maximizes the chances of finding good neighbors. Determining the facets of fundamental regions is particularly complicated for some lattices due to the complexity of the geometry of the Voronoi regions.

IV.3. Unstructured Quantization

We now move to describing unstructured quantization schemes. Unstructured quantization schemes take into account the statistics of data and produce cells that somehow nicely capture the distribution of data in space. Cells are identified by representative points, and there is typically no regularity in their locations and/or sizes when quantizing real world data sets. Fast decoding procedures relying on algebraic formulas thus do not exist, and some other methods must be used to organize efficiently the set of representative values.

An unstructured quantizer is typically built from the analysis of a (large) sample of the data that will eventually be indexed. The (Euclidean here) unstructured quantizer g is defined as a function

$$\begin{array}{rcl} \mathbb{R} & \rightarrow & [1, \ldots, k] \\ x & \rightarrow & g(x) = \arg\min_{i=1..k} L_2\left(x, c(i)\right) \end{array}$$

mapping an input vector x to a cell number $g(x)$. The integer k is the number of possible values of $g(x)$. The vectors $c(i), 1 \leq i \leq k$ are called *centroids* and suffice to define the quantizer.

At indexing time, the data sample is used to determine the k centroids. Then, all points from the data collections are typically assigned to the cells defined by these centroids. This assignment phase first determines for each point of the data collection its closest centroid and then stores in the associated cell the point. Indexing is complete when all points have been assigned to the cell of their closest centroid.

At search time, the centroid $c(i)$ that is the closest to the query point is identified. The associated cell is fetched and its contents analyzed to discover the most similar vectors it contains.

IV.3.1. k-means

k-means is a very popular choice for constructing a good unstructured quantizer. In this case, k corresponds to the number of clusters. The k-means algorithm eventually builds Voronoi cells, each cell being identified by its centroid. k-means only finds a local minimum of the within-cluster sum of square distances.

The simplicity of k-means and its extremely good quantification properties have motivated researchers to extensively use this algorithm as a foundation for designing innovative indexing schemes. In 2003, Sivic and Zisserman proposed one extremely interesting approach based on a k-means unstructured quantization scheme [Sivic and Zisserman, 2003], called Video Google. They observed that text-based information retrieval engines work very well at very large scale, both in terms of efficiency and effectiveness. They proposed a scheme to create, in the visual domain, the notion of words that is so familiar with text-based search engines. To this end, they quantized using a k-means algorithm a large set of SIFT descriptors to learn 20,000 centroids. Once the centroids learned from the sample, they then assigned to these centroids the remaining descriptors of the collection to index. Each centroid defines a visual word, the centroids a visual vocabulary, and two descriptors get quantized to the same visual word if they are similar enough. In [Sivic and Zisserman, 2003], all the descriptors quantized to the same Voronoi cell form a bag,

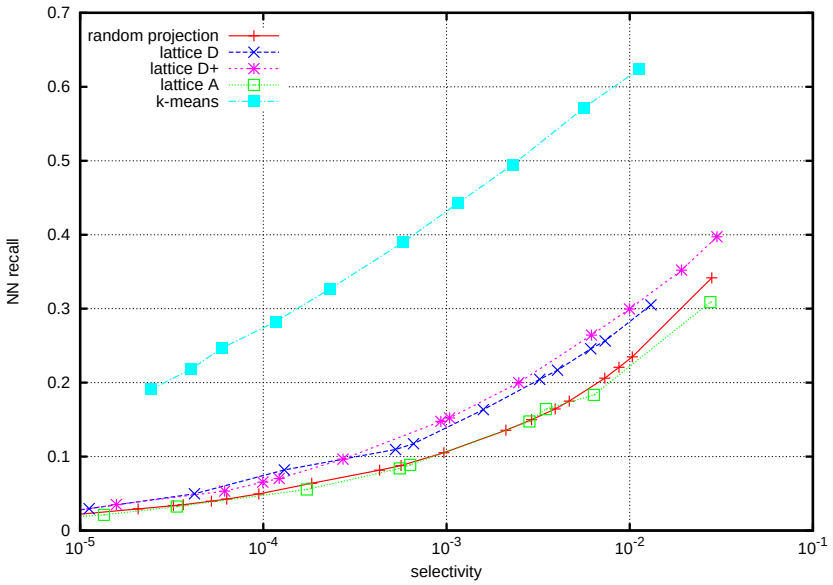


Figure IV.3.: Evaluating the performance of various quantization schemes with LSH

i.e., no particular order constrains their layout. For that reason, this approach is often termed "Bag-of-Features, BoF". They also proposed a inverted-list indexing scheme to quickly identify the images from the indexed collection that are similar to the query image.

From a bird's eyes view, this approach can be split into three phases:

1. *Learning Phase:* a k-means is ran to learn the visual vocabulary. This is done by clustering a large sample of the data collection (typically few million descriptors) into a relatively small numbers of centroids (typically few tenth of thousands). This phase needs to determine the closest centroid for every descriptor of the sample and every k-means iteration. Convergence is rather quick and typically reached after a couple of dozen iterations. This phase is CPU intensive. Once convergence has been reached, then the set of centroids defines the visual vocabulary.

2. *Assignment Phase:* Once the visual vocabulary has been determined from the sample, then all descriptors from the collection to index are assigned to their closest centroid. To this end, the distances between the current descriptor and all centroids are computed and the shorter distance gives the centroid to assign the descriptor to. This phase is CPU intensive. Overall, this phase enables to associate to each image a vector of visual word occurrences.

3. *Search Phase:* The vector of visual word occurrences is built from the descriptors of the query image. The most similar vectors of occurrences are identified using an inverted-list data structure, with some *tf-idf* weighting scheme before returning the most similar images, ranked. This phase is cheap in comparison, but it is consuming quite a lot of memory as it needs to maintain a representation of the (rather sparse though) matrix of word occurrences for the entire indexed image collection.

IV.3.2. Comparing Approaches

Unstructured quantization schemes proved to work extremely well in practice, much better than structured quantizers. To demonstrate this nice behavior, we ran a little study comparing the performance of the LSH approach [Gionis et al., 1999] when the quantization functions are on the one hand the traditional random projections and on the other hand when they are replaced by vectorial quantizers that are structured or unstructured. We implemented three Lattice based quantizers replacing the random projections, D, D^+ and A. We also replaced LSH's hash-functions with a k-means quantization scheme.

For this study, we used a vector dataset extracted from the publicly available INRIA Holidays dataset[3], which is composed of high-definition real holidays photos. There are many series of images with a large variety of scene types (natural, man-made, water and fire effects, etc). Each series contains somehow visually similar images, differing however due to various rotations, viewpoint and illumination changes of the same scenes taken by the photographers.

These images have been described with SIFT. The descriptor collection used in this experiment is a subsample of 1 million descriptors randomly picked from the descriptors of the image dataset. We also randomly picked 10,000 descriptors used as queries, and another set of 1 million vectors extracted from a distinct image dataset (downloaded from Flickr) for learning the centroids in the case of the k-means based approach. Finally, we ran exact searches using the Euclidean distance to get the true nearest neighbor of each of these query descriptors—this builds the ground truth against which the various instanciations of LSH will be compared.

Figure IV.3 gives the evaluation of the different types of quantization schemes used in an LSH setting. The x-axis is the selectivity, i.e., the fraction of the data collection that is read at search time and from which the result is eventually built. Selectivity is a key factor when considering a database perspective, as it basically rules the number of I/Os. In practice, traveling along the x-axis turns into varying the number of quantization cells that are analyzed at search time. The y-axis gives the nearest neighbor recall, i.e., the proportion of the points that belong to the pre-computed ground truth and that are part of the result that is returned.

For both random projection and lattices, the two parameters driving the quantization step w and the number of dimensions d^* picked to quantize the data are optimized. Figure IV.3 only presents the optimal ones, which are obtained as follows. Given a pair of values for the parameters w and d^*, we compute the nearest neighbor recall at a given selectivity. This process is repeated for a set of varying pairs for these two parameters, resulting in a set of tuples associating a selectivity to a nearest neighbor recall. Points plotted on the curves belong to the roof of the convex hull of these numbers. Therefore, a point on the figure corresponds to an optimal parameter setting, the one that gives the best performance obtained for a given selectivity. For the k-means hash function, only one parameter has to be fixed: the number of centroids k, which gives the trade-off between recall and selectivity. This simpler parametrization is an advantage in practice.

Figure IV.3 clearly shows that the lattice quantizers provide significantly better results than random projections, due to the vectorial gain. These results confirm that the random projections used in LSH are unable to exploit the spatial consistency. Note that this phenomenon was underlined in [Andoni and Indyk, 2006, Jégou et al., 2008b]. However, by contrast to these works, the lattices we are evaluating are more flexible, as they are defined for any value of d^*. In particular, the lattice E_8 used in [Jégou et al., 2008b] is a special case of the D^+ lattice.

Figure IV.3 also shows that the various types of lattice perform differently. We observe an improvement of the nearest neighbor recall with lattices D and D^+ compared to random

[3] http://lear.inrialpes.fr/people/jegou/data.php

projections, whereas lattice A gives similar performance. The density of D^+ is known to be twice the density of D. In high-dimensions, the density of A is small compared to that of D. Overall, density clearly affects the performance of lattices. However, density is not the only crucial parameter. The shape of the fundamental region and its orientation may also be influential, depending on the distribution of the dataset.

On the Figure IV.3, it is possible to observe the better performance of the k-means hash function design in terms of the trade-off between recall and selectivity. For the sake of fairness, the codebook (i.e., the centroids) has been learned on a distinct set. The improvement obtained by using this hash function construction method is very significant: the selectivity is about two order of magnitude smaller for the same recall.

IV.3.3. Unstructured Quantization: Problems and Extensions

The seminal Video Google approach to image retrieval uses k-means to group descriptors into visual words, which are then indexed using information retrieval techniques; in this case, the contents of the clusters is not used directly for query processing [Sivic and Zisserman, 2003]. Rather, what matters is the fact that a point is assigned to a cluster. Soon after the scheme proposed by Sivic and Zisserman was published, some researchers have proposed extensions to fix the various problems this indexing approach has when scaling up the size of the data set.

Enlarging the Visual Vocabulary

One of the first extension was to use a k-means with a value for k that is much larger than the one used in [Sivic and Zisserman, 2003]. Nistér and Stewénius observed that the retrieval quality was increased when the visual vocabulary is significantly enlarged to several million centroids [Nistér and Stewénius, 2006]. Increasing the size of the vocabulary reduces the fraction of descriptors in each cluster. It makes the vector of visual word occurrences more sparse, which is beneficial for performance.

Historically, however, the value of k when running the k-means algorithm for indexing images has been quite small, few tenths of thousands. Keeping k small was essentially motivated by the otherwise high cost of the learning phase since it requires performing an enormous amount of distance calculations. Having a rather small visual vocabulary hence reduces the costs of the learning and the assignment phases. But it produces results that are of poor quality and that are slow to obtain. The poor quality is related to the quite large quantization errors that are made for each visual word. Indeed, rather dissimilar descriptors end-up being assigned to the same centroid, which tend to eventually produce false positives. Results are slow to obtain because a large fraction of the database must be analyzed as each visual word indeed clusters many points. These two reasons motivate the need to enlarge the vocabulary, increasing sparsity, thus improving performance.

On the one hand, increasing the numbers of centroids is desirable, on the other, care must be taken to keep low the costs of the learning and assignment phases. For that reason, in parallel to increasing the number of centroids, building a hierarchy of centroids and indexing them was proposed as to reduce the cost of identifying the representative that is the closest to the data point under scrutiny [Philbin et al., 2007, Nistér and Stewénius, 2006].

The observation by Nistér and Stewénius [Nistér and Stewénius, 2006] materializes as follows. They propose an approximate version of a hierarchical k-means based quantization scheme. Their approach is to recursively computes a k-means, with k relatively small for the internal nodes until obtaining a pre-defined tree height. This produces a balanced tree structure, where

each internal node is connected to a fixed number of centroids. The search is performed top-down by recursively finding the nearest centroid until a leaf is reached. The method uses two parameters, the height of the tree h_{t} and the branching factor b_{f}. The total number of centroids (leaves) is then obtained as $(b_{\mathrm{f}})^{h_{\mathrm{t}}}$.

Accelerating the clustering of the data collection in the Video Google context is also the goal of Philbin et al. [Philbin et al., 2007]. Their clustering process is flat, similar to standard k-means. They basically reduce the number of representatives each point must be compared to, boosting the assignment and trading-off speed for accuracy. They start by precomputing a large set of cluster representatives that get indexed into several randomized K-D-trees. They assign a data point to its approximate closest representative by first probing each K-D-tree with the point to cluster.

A Note on Product Quantization, the PQ-Code Scheme. Vectorial quantization works best when the quantization noise is extremely small, that is, when the representatives truly capture the location of points in the high-dimensional space. Minimizing the quantization error is possible by significantly increasing the number of representatives. This is indeed what Nistér and Stewénius observed [Nistér and Stewénius, 2006]. None of the extensions to k-means described above can work with as many as 2^{64} representatives. There would never be enough data to learn the dictionary, the complexity of the algorithm would be too large and no computer would have enough memory to store even the coordinates of the high-dimensional centroids.

Product quantization is an efficient solution to address this issue [Jégou et al., 2011]. Product quantization is a common technique in source coding [Gray and Neuhoff, 1998]. Instead of quantizing the space using a very large number of k representatives, quantization proceeds as follows. The high-dimensional vector $x \in \mathbb{R}^d$ to quantize is first split into m distinct subvectors of dimension $d' = d/m$. Each subvector[4] is quantized separately using k' representative values. Overall, m such quantizers are used, each defining a sub-codebook. The overall codebook of the entire product quantization scheme is the Cartesian product of all the sub-codebooks. Therefore, product quantization encodes vectors using a $k = (k')^m$ codebook but needs to maintain only m different codebooks of k' centroids. The strength of a product quantization is to produce a large set of centroids from several small sets of centroids. Complexity is therefore related to k' and m and not to k, making the corresponding PQ-Code high-dimensional indexing scheme built on this product quantization ideas very useful in practice.

This extremely smart vectorial quantization scheme is from my point of view the best contribution found in a publication since many years.

Soft Assignment

In [Philbin et al., 2008], Philbin et al. demonstrate that assigning one point to more than one cluster enhances the quality of the result. This is called soft-assignment, and this strategy has been applied by [Nistér and Stewénius, 2006], among others. Note that this is somehow dual to the notion of multi-probe we discussed in Section III.2.3.

Addressing Disks Issues

Nistér and Stewénius subsequently considered the need to have clustered data kept on disks. They therefore designed a k-means type-of technique to do vectorial quantization where the

[4]For best performance, each subvector should roughly carry the same energy.

quantized data is stored on hard disk drives. Their scheme uses simultaneously multiple (15–20) different clusterings to ensure quality, requiring one disk I/O per cluster for each query descriptor [Fraundorfer et al., 2007]. This is clearly not an option from a database perspective, doing so many I/Os would be way too costly.

Optimizing for CPU Consumption

With the Cluster Pruning algorithm [Chierichetti et al., 2007], Chierichetti et al. addressed the specific problem of reducing the CPU load when constructing the quantizer as well as when using the index at search time. Their approach, however, purposely ignores disks costs. Ignoring disks, it is best to cluster a collection of n descriptors into $\sqrt{n}$ clusters containing, on average, $\sqrt{n}$ descriptors each; this minimizes the total CPU cost of the retrieval. The clustering process they use is extremely simple: they randomly pick a given number of representatives from the collection and then assign all remaining points to these representatives. Unlike the traditional k-means approach, there are no iteration here, no convergence, there is a single phase of assignment. There is therefore no attempt to reach any local optimum for the intra-cluster distances.

For large collections, $\sqrt{n}$ is a large number, resulting in excessive CPU cost. To reduce that cost, Chierichetti et al. propose to hierarchically cluster the set of representatives. Their parameter L controls the number of levels in the hierarchy. This is somehow similar to the hierarchical k-means [Nistér and Stewénius, 2006], but again, it differs because no iterations are performed to find a local optimum for centroids.

IV.4. Vectorial Quantization from a DB Perspective

This section discusses the problems vectorial quantization schemes face when observed from a database perspective. It also proposes two mechanisms making such schemes more I/O friendly. This section finally proposes a study where various secondary storage technologies are used in order to understand what is their respective impact on the performance of indexing techniques. This is particularly important as new storage techniques emerge, slowly replacing magnetic disks and profoundly changing the perspective on I/O bottlenecks.

Before entering this discussion, however, it is important to make the following remark. The extremely successful approaches relying on a BoF type-of approach perform very well because they do image recognition at the level of entire images. The BoF model is extremely inaccurate, it simply considers the assignments, ignoring the contents of the clusters. BoF type-of approaches can not do as fine grain image recognition as other techniques where individual local descriptors are preserved, or where actual distances are computed. It is crucial to note BoF type-of approaches hard-wire the use of local descriptors, they can not be used with global approaches. This somehow clashes with the genericity perspective from Section I.4.

IV.4.1. Lessons from the Literature and DB-Oriented Motivations

The literature tells us unstructured vectorial quantization schemes should be used, as they much better perform than any other scheme, whether it is scalar or vectorial but structured. The k-means algorithm is often used as a vectorial unstructured quantizer for high-dimensional indexing because it has nice properties: it is a simple algorithm, surprisingly effective and easy to implement; it nicely deals with the true distribution of data in space by minimizing the mean square error over the clustered data collection.

When observed from a database perspective, then several problems arise. They are, again, typically associated with scale, as large data sets worsen the impact of any bottleneck. Several problems must be tackled when using a k-means type-of algorithm for designing a high-dimensional indexing solution that remain efficient at large scale and usable in practice when data is stored on disks. The first problem is related to the great variation in the cardinalities of the clusters that are produced by k-means. The second problem is the lack of understanding of disk-based clustering algorithms for high-dimensional indexing. The third problem is related to the use of new technologies for storing data, which can potentially change the design and the implementation of indexing schemes. We discuss these problems below.

Clusters are Severely Imbalanced

k-means tends to produce clusters having quite different cardinalities. This, in turn, impacts the performance of the retrieval algorithm when it needs to process the contents of the clusters: scanning heavily filled clusters is costly as the distances to many points must be computed. In contrast, under-filled clusters are cheap to process, but they are selected less often as the query descriptors are also less likely to be associated with these less populated clusters. Overall, having imbalanced clusters impacts both the variance and the expectation of query response times. This is very detrimental to contexts in which performance is paramount, such as high-throughput settings since the true resource consumption can no more be accurately predicted by costs models. This phenomenon has an even more detrimental impact at large scale since clusters must be stored on disks and the performance severely suffers when fetching large clusters due to the large I/Os. Furthermore, k-means is known to fail clustering at very large scale, and hierarchical or approximate k-means must be used, which, in turn, tend to increase the imbalance between clusters [Jégou et al., 2010b]. It is therefore needed to come up with a mechanism that produces clusters of a much more even size. We describe such a technique below in Section IV.4.2.

k-means Has to Go to Disks

The bulk of the work using a k-means approach as a basis for high-dimensional indexing addresses main memory settings. Few techniques addressing secondary storage issues have been published, notably of [Fraundorfer et al., 2007]. In this work, however, little investigations were done to understand how to best treat secondary storage to maximize the performance. Disk were mostly added to the existing scheme, as an afterthought. We present next in Section IV.4.3 a study where we extend the Cluster Pruning [Chierichetti et al., 2007] approach in order to gracefully address disks issues.

From Magnetic to Solid State Disk Storage Units

Recent contributions from the database community make the case for getting rid of magnetic hard disk drives and use instead Solid State Disks, SSDs [Gal and Toledo, 2005, Lee et al., 2008]. SSDs have emerged as a disruptive storage technique based on memory cells on chips. Their storage capacity grows quickly and they dramatically outperform all magnetic approaches. It is therefore of high interest to study what impact secondary storage technologies can have on the design and performance of k-means type-of unstructured vectorial quantization schemes at the core of many high-dimensional indexing algorithms. We report the conclusions of such a study Section IV.4.3.

IV.4.2. Balancing Clusters to Reduce Response Time Variance

This section proposes an extension to the traditional k-means algorithm to produce clusters of much more even size [Tavenard et al., 2011]. This is beneficial to performance since it reduces the variance and the expectation of query response times. Balancing is obtained by slightly distorting the boundaries of clusters. This, in turn, impacts (slightly negatively) the quality of results since clusters do not match the initial optimization criterion anymore.

A k-means quantizes N high-dimensional feature vectors into k clusters defining Voronoi regions. This process intends to minimize the mean-squared-error (MSE) criterion, that is:

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c_i}\|^2,$$

where C_i is the i-th cluster and $\mathbf{c_i}$ is its associated centroid.

Without loss of generality, each cluster stores a list of its associated data points, as it can be read in the following papers [Sivic and Zisserman, 2003, Nistér and Stewénius, 2006, Douze et al., 2009, Jégou et al., 2010a]. At query time, one or a few clusters are probed. It is known that the quality of results is increased when probing multiple clusters [Lv et al., 2007, Joly and Buisson, 2008, Jégou et al., 2010b, Jégou et al., 2010a]. In general, the actual distances between the query point and the features stored in each cluster relevant for that query are subsequently computed [Datar et al., 2004, Muja and Lowe, 2009]. Therefore, the response time of a query is directly related to (i) the strategy used to identify the clusters to probe, and (ii) the total number of vectors used in distance computations.

The cost for (i) is fixed and mainly corresponds to finding the m_p centroids that are the L_2-closest to the query point. In contrast, the cost for (ii) heavily depends on the cardinality of each cluster to process. It is of course linked to m_p. Note that (i) is often negligible compared to (ii).

Probing m_p clusters results in a list of vector candidates for which distances to the query vector must be computed. The trade-off between result quality and retrieval time is measured by *recall* and *selectivity*, respectively. *Selectivity* is the fraction of the data collection that is used in the distance calculations. Obviously, the larger selectivity, the higher the cost of the retrieval process. Selectivity is very important from a database perspective as it translates to costly I/Os. *Recall* is the average rate of nearest neighbors returned for a query, with respect to a given ground truth. In other words, recall is equal to the number of true results returned among the total set of true results. Observe that if the true nearest neighbor is found within any of the selected clusters, then it will be ranked first after having performed exact distance computations [Datar et al., 2004, Muja and Lowe, 2009].

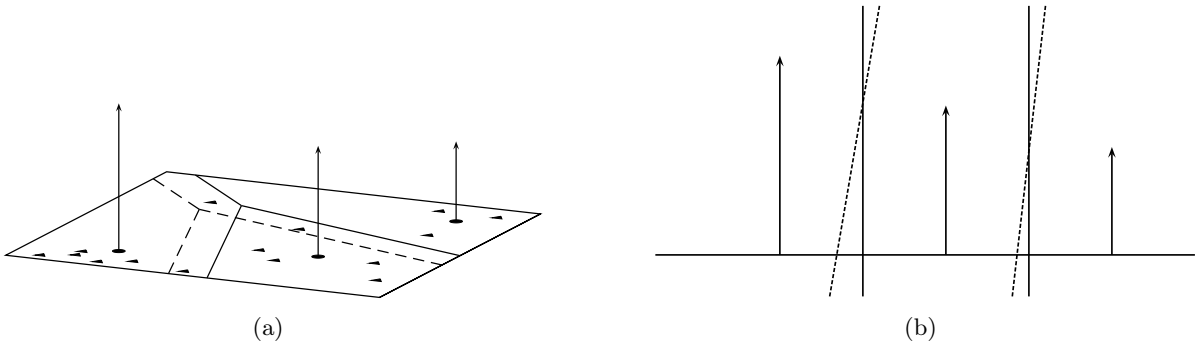
Defining an Imbalance Factor

As in [Jégou et al., 2010b], we empirically measure the imbalance between the cardinalities of the clusters resulting from a k-means using an *imbalance factor* γ defined as:

$$\gamma = k \sum_{i=1}^{k} {p_i}^2$$

where p_i is the probability that a given vector is stored in the list associated with the i^{th} cluster.

As shown in [Jégou et al., 2010b], for a given k and $m_\mathrm{p} = 1$, γ is directly related to the search cost: $\gamma = 3$ means that the expectation of the search time is three times higher than the



(a) (b)

Figure IV.4.: Illustrating the balancing of clusters. (a): 3-d embedded data points and centroids. (b): Voronoi region boundaries shifted after some iterations

one associated with perfectly balanced clusters. This is the reason why imbalance factor is, to our opinion, more appropriate to assess balancing than entropy, even if both metrics reach an extremum in the case of perfectly balanced clusters.

Optimal balancing ($\gamma = 1$) is obtained when $|C_i| = N/k$ for all i. In that case, the variance of query time is zero, as any given cluster contains exactly the same number of elements, as shown by the analytical expression of the variance of the number of elements in a cluster list:

$$\mathrm{Var} = N^2 \sum_{i=1}^{k} p_i \left(p_i - \frac{1}{k} \right)^2 .$$

Balancing Clusters

Balancing clusters is an iterative post-processing step performed on the final output of a k-means type-of algorithm. The idea is to artificially enlarge the distances between the data points and the centroids of the heavily filled clusters so as to shrink and slightly drain loaded clusters. Penalties applied to distances depend on the population of clusters. Hence, the contents of clusters and thus their population can be recomputed accordingly. This balancing process eventually converges to equally filled clusters. The penalties are called *penalization terms*, denoted by b_i^l when penalizing cluster i at iteration l and are computed as follows:

$$\begin{cases} \forall i, b_i^0 = 1 \\ \forall l \leq r, b_i^{l+1} = b_i^l \left(\frac{n_i^l}{n_{\mathrm{opt}}} \right)^{\alpha} \end{cases} \tag{IV.1}$$

where α controls the convergence speed and r is the number of iterations performed. A small value for α ensures that balancing will be done in a smooth way. A small α however implies to have greater r in order to get clusters of even population. Note that, at each iteration l, the populations $|C_i^l|$ are updated so as to take these penalization terms into account. More precisely, distances from any point $\mathbf{x}$ to the i^{th} centroid are set to

$$d_{\mathrm{bal}}^l(\mathbf{x}, \mathbf{c_i})^2 = d(\mathbf{x}, \mathbf{c_i})^2 + b_i^l.$$

The initial value for b_i^0 assigns an energy to balance factors that is comparable to the one of other dimensions of the data, under the assumption that vectors are normalized.

The proposed balancing strategy empirically converges towards clusters having the same cardinalities. Several stopping criteria can be applied, the most simple being a maximum number

descriptor	dimensionality	γ	
		k=256	k=1024
SIFT	128	1.08	1.09
BoF	1000	1.65	1.93
GIST	960	1.72	3.75
VLAD	8192	5.41	6.23

Table IV.1.: Imbalance factor for k-means computed on 10^6 images descriptors of different types

of iterations. It is also possible to target a particular value for γ, either fixed or possibly in proportion to the initial imbalance factor. Early stopping the balancing process reduces the overall distortion of the initial Voronoi regions.

Geometrical Interpretation

A geometrical interpretation of the balancing process described above is possible. Assume the balancing process first embeds the k-means clustered d-dimensional vectors into a $(d+1)$-dimensional space. In this new space, the d first components of all vectors are the ones they had in their initial space, while component $d+1$ is set to zero. Centroids are embedded similarly, except for their last component that is set to $\sqrt{b_i^0}$. Then, along iterations, that last component is set to the appropriate $\sqrt{b_i^l}$ value. The intuition is that centroids are elevated in an iterative way from the hyperplane where vectors lie. The more vectors in a cluster, the more elevation its centroid gets.

This is illustrated in Figure IV.4(a), where the z-axis corresponds to the added dimension. In this figure, data points and centroids are embedded in a 3-d example. Data points are plotted as triangles while centroids are represented as dots, with a non-null z-axis value after some iterations. Initial Voronoi region boundaries and their shifted version are respectively plotted as solid and dashed lines. While iterating, updated assignments for vectors are computed with respect to the coordinates of the points lying in the augmented space. The artificial elevation of centroids tends to shrink the most populated clusters, dispatching some of its points in neighboring clusters.

Figure IV.4(b) exhibits the influence of the $(d+1)^{\text{th}}$ coordinate of the centroids on the position of the borders. Voronoi region boundaries shifted after some iterations. Along iterations, clusters of large population get their centroids moved away while other centroids stay close to the $z=0$ plane where data points lie. New boundaries, plotted as dashed lines, shrink the left-hand-side cluster because of its large population.

Note that new Voronoi region boundaries are parallel to initial ones at each step, which can be shown as follows. Updated boundary between clusters i and j is defined as

$$H_{i,j} = \left\{ \mathbf{x}, f(\mathbf{x}) + b_i^l - b_j^l = 0 \right\}, \text{using } f(\mathbf{x}) = d(\mathbf{x}, \mathbf{c_i})^2 - d(\mathbf{x}, \mathbf{c_j})^2$$

where b_i^l and b_j^l are constant with respect to $\mathbf{x}$. This implies that the vector $\left(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_d}\right)$ is normal to $H_{i,j}$. As it is also normal to the initial boundary, both boundaries are parallel.

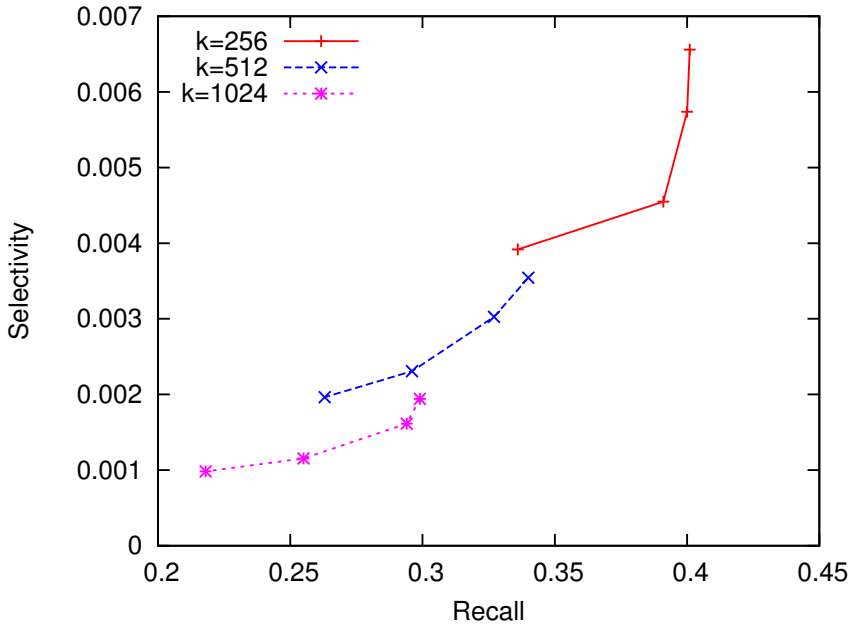


Figure IV.5.: Impact of varying balancing on the trade-off selectivity/recall. The 4 points along the 3 curves each correspond to 0, 4, 16 and 64 iterations, *top to bottom*

Experiments: Datasets and Imbalance Factor Analysis

We now experimentally illustrate the resulting improvements. Our analysis has been performed on descriptors extracted from a large set of real-world images. We downloaded one million images from Flickr to build the database and another one thousand images for the queries. Several description schemes were applied to these images, namely SIFT local descriptors [Lowe, 2004], Bag-of-features [Sivic and Zisserman, 2003] (BoF), GIST [Oliva and Torralba, 2001] and VLAD descriptors [Jégou et al., 2010a]. SIFT were extracted from Hessian-Affine regions, according to the approach proposed by Mikolajczyk and Schmid [Mikolajczyk and Schmid, 2004]. BoF vectors have been generated from these local descriptors, using a codebook obtained by regular k-means clustering with 1000 visual words. The VLAD descriptors were generated using a codebook of 64 visual words applied to the same SIFT descriptors, leading to vectors of dimension $64 \times 128 = 8192$. For GIST, we have used the most common setup (colors and 3 scales), leading to 960-dimensional descriptors. Global descriptors (BoF, GIST and VLAD) produce exactly one descriptor per image, leading to one million vectors for each type of descriptor. We have randomly sampled the SIFT local descriptors to keep 1 million vectors, so as to allow fair comparison between γ values for all types of descriptors.

Table IV.1 reports the observed imbalance factors obtained for each type of descriptors after performing a standard k-means clustering on our database. It can be observed that higher dimensional vectors tend to produce higher imbalance factors. BoF descriptors have an imbalance factor which is lower than GIST for a comparable dimension, which might be due to their higher sparsity. The number k of clusters has a significant impact on γ: larger values for k lead to higher γ. The low values for k we have considered here explains why γ measured for the SIFT descriptors in Table IV.1 are lower than those reported in the literature: [Jégou et al., 2010b] reports 1.21 and 1.34 for codebooks of size k=20,000 and k=200,000, respectively. Note that, in the case of hierarchical clustering, observed imbalance factor are much higher since balancing

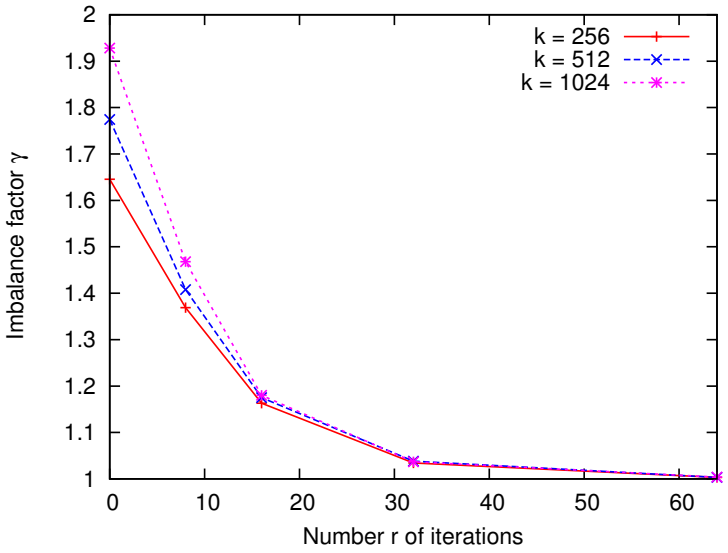


Figure IV.6.: Convergence speed in terms of the number r of iterations

issues are present at every level of the clustering tree.

We analyze here the impact of our method on selectivity, recall and variability of the response time. So as to better study the impact of clustering in the feature space, we use a ground truth based on actual feature space distances rather than on semantic image similarity. We also analyze the convergence properties of our method. α is set to 0.01 in all our experiments. Note that our balancing strategy is especially interesting for global descriptors where having perfectly balanced clusters leads to constant query time. Therefore we performed our experimental validation on the well known BoF vectors. In the following, we use $m_{\mathrm{p}} = 1$ in order to better show the impact of balancing. Note however that significantly better recall performances could be achieved by using larger m_{p}. (m_{p} clusters are scrutinized at query time.)

Selectivity/Recall Performance. Figure IV.5 shows the trade-off between accuracy and complexity for different values of k. Observe that first iterations of our algorithm tend to significantly lower selectivity without severely impacting recall. Note also that the trade-off between selectivity and recall can be adjusted using the number k of clusters (and the number m_{p} of probes). Our method exhibits comparable performance with that of the k-means clustering in terms of selectivity and recall. For example, using $k = 512$, performing a k-means without balancing ($\gamma = 1.77$) leads to a selectivity of 0.0035 for a recall of 0.34, while performing only 4 iterations of our balancing strategy ($\gamma = 1.53$) allows one to achieve a selectivity of 0.0030 (-14%) for a recall of 0.33 (-3%). Note however that with our method a given selectivity/recall point is obtained with a much better (lower) variability of the response time, as shown later.

Impact of the Number of Iterations. The number r of iterations in Equation IV.1 is an important parameter of our method, as it controls to which extent complete balancing is enforced or not. Figure IV.5 shows that selectivity is reduced in the first iterations with a reasonable loss in recall, i.e., comparable to what we would obtain by modifying the number of clusters. Following iterations are comparatively less profitable, as the gain in selectivity is obtained at a relatively high penalty in recall. Modifying the stopping criterion allows our method to attain a target imbalance factor which is competitive with respect to the selectivity/recall trade-off.

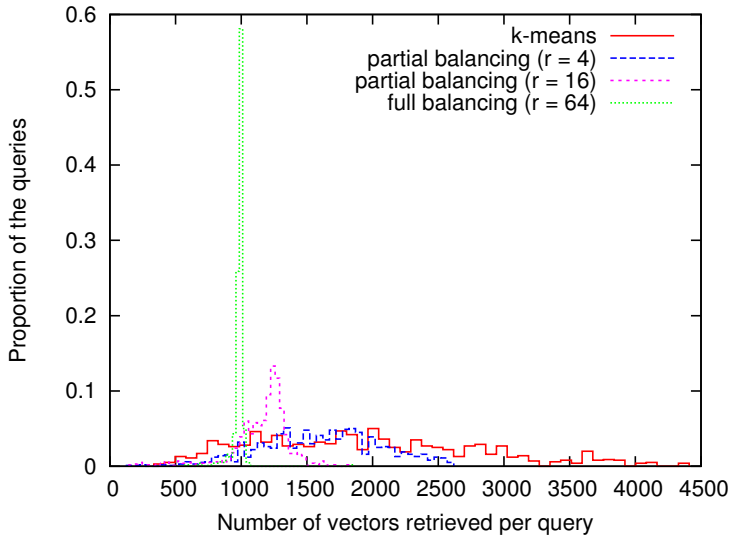


Figure IV.7.: Histograms of the number of vectors returned. 1000 queries, k-means and cluster balancing, three iterations, $k = 1024$ for all cases

Convergence Speed. Figure IV.6 shows that even a small number of iterations can lead to reasonably balanced clusters. Full balancing is almost achieved after 64 iterations, which leads to a computational cost that is negligible in the experiments we conducted compared to that of the clustering. Higher values of k for the initial k-means do not require more balancing iterations, which is somewhat surprising as more penalization terms have to be learned. Note that convergence of our algorithm is not guaranteed, though it has been observed in our experiments.

Variance of the Query Response Time. The impact of our balancing strategy on the variability of the response time is illustrated by Figure IV.7, which gives the distribution of the number of vectors used to compute distances and that are returned by the index for each query. Observe the tightness of the distribution in the case of our method, which reflects a very low variability in response time. The tight distribution obtained by our method shows that the objective of reducing the variability of the query time resulting from unbalanced clusters is fulfilled: the response time is almost constant with full balancing, while its variance is significantly lower with partial balancing than it is with the original clustering.

IV.4.3. eCP: a Disk Aware Vectorial Quantization Scheme

We briefly discussed page 63 the Cluster Pruning approach that was proposed as a simple way to cluster the data points [Chierichetti et al., 2007]. While the original evaluation of the algorithm was performed within a text indexing context at a small scale, its simplicity motivated us to study its behavior in an image indexing context at a much larger scale.

Cluster Pruning has been designed for in-memory settings. Dealing with larger scales make disk I/Os compulsory. This breaks some of the assumptions that are fundamental to Cluster Pruning. We therefore describe here several extensions that dramatically improve its performance and scalability, accelerating both query processing and the construction of clusters. The resulting algorithm is called extended Cluster Pruning, or eCP.

Extension #1: Creating a Hierarchy of Representatives Beforehand

The original Cluster Pruning algorithm works as follows. When N points are in the collection, then it randomly picks $\sqrt{n}$ points that are temporarily used as representatives. These points are called *leaders*. Then, the entire data collection is compared to all the leaders and each point assigned to its closest leader. Finally, once the clusters have been formed, a definitive cluster *representative* is chosen, per cluster. The obvious choices are the cluster leader itself, the centroid of the cluster, or the medoid of the cluster. At query time, the query point q is first compared to the set of the definitive $\sqrt{n}$ representatives to find the nearest one. Then, q is compared to all the points in that representative's cluster, to find its k nearest neighbors.

By construction, Cluster Pruning creates a tree of definitive representatives *only once* the assignments are over. This is because Chierichetti et al. decide between using random points, centroids or medoids as definitive representatives only after having completed the assignments. Therefore, no tree can be built to accelerate the assignments since the eventual representatives are not known until the actual clusters have been created.

In contrast, we have no other choice than to create a tree of representatives before starting the assignments if we want to scale to very large collections. We therefore propose to use random points as definitive representatives. It is then possible to create a top-down index of the representatives before starting assigning. Subsequently assigning points to clusters can therefore benefit from that index, dramatically reducing clustering time. This optimization is not possible with the other choices of cluster representatives, as those are known only once the collection clustered. To increase the quality of the index tree, we apply a soft assignment technique [Philbin et al., 2008] where a representative at one level is assigned to three representatives at the parent level.

While this setting does increase the index size, it can still easily fit in memory. This soft assignment applies only to cluster representatives in the index tree and does not apply to data in clusters. This, somehow, compensates for the "brutal" assignment of points to clusters. Finally, note that the scale of the database precludes applying soft assignment at the level of the clusters themselves, otherwise resulting in data explosion.

Extension #2: Cluster Size Selection

The results reported in [Sigurðardottir et al., 2005] indicated that cluster size is a key factor in the performance of cluster indexing, and that cluster size should be heavily influenced by the characteristics of the hard disk drive that descriptors reside on. In the original Cluster Pruning approach, however, there is a large difference in cluster sizes when using a single- or a two-level index, and both are independent of the I/O granularity of the disk. While this behavior minimizes the CPU cost, increasing the number of levels leads to very small descriptor clusters on disk, which under-utilize the I/Os, and a correspondingly large index.

We propose to explicitly set the value of the desired average cluster size and then determine the number of representatives as follows:

$$l = \left\lceil \frac{number\ of\ descriptors}{\lfloor desired\ cluster\ size\ /\ descriptor\ size \rfloor} \right\rceil$$

where the value of *desired cluster size* should match the operating system I/O granule size for best performance. Using this new number of cluster representatives, the clustering proceeds as before. When the number of levels $L > 1$, each intermediate-level representative still represents $\sqrt[L]{l}$ points at the next level.

By decoupling the size of the clusters from the choice of L, we gain two major benefits. First, larger clusters lead to a smaller index that may fit entirely in memory. Second, as each cluster is larger, fewer clusters may potentially be read. While CPU cost is sacrificed, the I/O cost is reduced resulting in lower overall query processing cost.

Extension #3: Balanced Size Distribution

We know from the observation made previously in this chapter as well as by the contribution presented Section IV.4.2 that clusters are imbalanced. This is also true with Cluster Pruning. Instead of applying the method described above, we tried a very simple yet surprisingly effective method to balance the size distribution. We intentionally choose $X\%$ additional cluster representatives in the initial step of the algorithm. At the end of the cluster creation process, we then eliminate the corresponding number of the smallest cluster representatives by reclustering their descriptors into the l remaining clusters. In addition to the obvious advantage of eliminating the smallest clusters, the choice of additional representatives reduces the size of the largest clusters as the representatives now better represent the descriptor distribution.

We have chosen not to recluster the largest clusters. The reason is that since large clusters typically occur in dense areas of the descriptor space, it is likely that reclustering a large cluster would simply move all the descriptors to a single cluster (or a few), resulting in that cluster becoming equally large as the removed one, or even larger.

We have not compared this balancing strategy and the one presented above in Section IV.4.2.

Summary of Performance Experiments

We measured the performance of the eCP approach using various data sets, including the ones presented Section II.4. On the one hand, we carefully evaluated the quality of the indexing scheme by checking the results against the ground truth. On the other, we took time measurements for assessing the performance of the indexing phase as well as the search phase. Of course we logged many additional indicators to provide figures for cache hits, number of I/Os, etc.

The following summarizes the outcome of various experiments with eCP, published in several venues [Guðmundsson et al., 2010, Moise et al., 2013, Shestakov et al., 2013]. The goal here is not to repeat what these publications contain, times and numbers. The goal of this summary is rather to give to the reader the key elements that really matter for designing a disk-aware vectorial unstructured quantization scheme.

We start with quality, as it is of course central to make sure eCP returns good results at scale. We then move to discussing other facets of eCP's behavior.

eCP Meets Result Quality Standards. In [Moise et al., 2013, Shestakov et al., 2013], we have checked the quality of the results returned by eCP when using a set of queries searching for quasi-copies lost in the middle of 100 million distracting images, or about 30 billion SIFT. This data set corresponds to the one described in details in the Section II.4.3, and the query and ground truth used are the 49k and the Copydays sets described Sections II.4.1 and II.4.2. We made other experiments checking the quality of eCP, but this one is the most challenging as (i) the number and hence the diversity of the distracting images are high and (ii) the images themselves are small, generating few SIFT descriptors, making the matching quite difficult to establish. Please note that this experiment will be detailed more precisely in Chapter V.

Copydays contains 3055 variants of 127 original images. Variants were generated according to three families of visual modifications: jpeg-compressions (ranging from quality 75 to quality 03),

crops (removing 10% to 80% of the image, then rescaling) and strong modifications (manually generated distortions such as print-crumple-scan). Overall, 82.16% of the variants are found despite being drown into 100M distracting images. The variants that are occasionally not found correspond to crop-80% and some strong variants. Note that sometimes such variants create only a handful of SIFT descriptors, sometime no descriptor at all. Result quality when using the 49k dataset and the 100M distracting set is close to 100%, as the variants are easier to match.

Cluster Size Should Be Aligned with I/O Granule Size. We have been checking various sizes for creating clusters ranging from 32KB to 512KB. This, in turn, changes the number of clusters that must be created to fit the entire data collection.

As expected, having more (smaller) clusters result in a longer clustering process, as each descriptor must be compared to a greater number of representatives. In contrast, and again, as expected, scanning smaller clusters is most efficient since fewer distance calculations must be made.

There is no significant difference in the time it takes to read from disk clusters having their size smaller or equal to the I/O granule size (typically 128K); this is normal as the operating system always fetches data according to a granule-based policy. Processing a 32K cluster when 128K have been read under-utilizes the I/O. Larger clusters obviously ask the operating system to perform multiple I/Os.

Quality wise, the best results are obtained with clusters of 64–128KB. Smaller clusters tend not to contain the expected neighbors, larger clusters contain what we expect as well as a large amount of noise. We conclude that the best combination of clustering time, search performance, I/O profitability and result quality is achieved using an average cluster size of 128KB.

Multilevel Index is Compulsory at Scale. We indexed the collections varying L and measured the time taken for building the index as well as the time it takes to answer all queries. It is significantly faster to use a value for L greater than one as soon as the collection grows in size. Relying on a hierarchy to prune the comparison space is mandatory—simply changing the value of L from 1 to 2 when creating about 12,000 clusters divided the creation time by 20. This observation corroborates some of the critics made to the standard, non-hierarchical k-means algorithm.

At very large scale, relying on a multilevel index is compulsory. For example, the largest collection having 100 million distracting images requires to create about 31 million clusters to be aligned with the I/O granule size (128K). It would not be efficient at all to check every point of the collection against the 31 million representatives in order to find the closest one to get assigned to. Instead, a 5-level index would shrink down this number to about 400 distances to compute, on average, per point.

Balancing Clusters Improve Performance. As this was highlighted earlier in Section IV.4.2, balancing the clusters is beneficial for the efficiency of the indexing method as it reduces the response time variability when answering queries. This conclusion also applies here. The rather rough balancing technique tried for eCP is able to reduce the number of very small clusters as well as the cardinality of very large clusters. It is however not able to really balance clusters as nicely as does the more sophisticated technique presented above.

Disk	Type	Specified Ave. Seek Time	Specified Ave. Rot. Latency	Specified Cache Size	Measured Seq. R/W Thr.put.
Seagate	Magnetic	11.0 ms	4.16 ms	8 MB	46/40 MB/s
Fujitsu	Magnetic	11.5 ms	4.17 ms	16 MB	68/53 MB/s
SuperTalent	SSD	<1 ms	-	Unknown	124/34 MB/s
Intel	SSD	<1 ms	-	16 MB	220/66 MB/s

Table IV.2.: Key storage device performance indicators

Impact of Storage Technology on the Efficiency of eCP Index Creation

One of the leitmotiv in this manuscript is that multimedia data collections being now extremely large, algorithms performing content-based retrieval must deal with secondary storage. Magnetic disks have been around for decades, but their performance, aside from capacity, has not improved significantly. Better storage performance, and improved reliability, has been achieved, however, by grouping many disks together and striping data as in the Redundant Array of Independent Disks approach. Recently, Solid-State Disks (SSDs) have emerged as a disruptive storage technique based on memory cells on chips. Their storage capacity grows quickly and they outperform magnetic approaches.

We have analyzed the impact of various storage technologies on the performance of the index creation of eCP, which is the most disk-intensive phase, far more intensive than the search phase. Not only must the entire data collection be read from secondary storage during index creation, and then eventually written back to secondary storage again, but a gigantic number of CPU intensive distance calculations between vectors is also required to cluster them. High-dimensional indexes are typically created in a bulk manner: all vectors to index are known before the process starts and the index tree structure, as well as the bottom leafs of the tree, are all created in one go. From a traditional DBMS perspective, this process can be seen as being analogous to a sort-merge process with very CPU-intensive comparison function calls.

Implementation of the eCP Index Creation. The index creation starts by building the in-memory index tree by picking representatives from the raw collection. Then it allocates a buffer, called *in-buff*, for reading the raw data collection in large pieces. It then iterates through the raw collection via this buffer, filling it with many not-yet-indexed vectors. The tree is used to quickly identify the representative that is the closest to each vector in *in-buff*, representing the cluster that the vector must be assigned to. Once all vectors in *in-buff* have been processed, then the contents of the buffer is pushed to secondary storage as follows.

The buffer *in-buff* is sorted on increasing values of the representative identifiers. Then, *in-buff* is flushed into a new temporary file created on disk. It then reads another large piece from the raw collection into *in-buff* and continues. After having processed all vectors from the raw collection, all temporary files are merged using a typical secondary storage merging process.

In terms of access patterns, the assignments do large sequential reads (typically 128MB) to fill *in-buff* and large sequential writes when creating each temporary file. When creating the final file, many small random reads are needed to get data from all the temporary files and large sequential writes are done for the final file.

Running Experiments. Experiments were run on a Dell Precision T3400, 3GHz Intel E6400 dual core CPU with 6MB cache and 4GB RAM (only one core was used). For all disks we used

the `ext3` file system and Debian OS. We tested two magnetic disks: 3.5" Seagate Barracuda 7200.10 and 2.5" Fujitsu MHZ2160BJ. Both are 7200 rpm disks with similar seek time and rotational latency. We also used two SSDs: SuperTalent FTM28GL25H and Intel X-25M, type SSDSA2MH080G1GC. Finally, we used a NAS 3070 from NetApp. Table IV.2 provides more details on the drives. The three first columns are filled using vendor figures, while the last column shows sustained observed sequential read and write performance. Accurately measuring the performance of the NAS is much more complicated.

We checked two configurations. One uses a single drive: the file containing the raw collection, the temporary files, and the final cluster file were all stored on a single disk. In this case some reads and writes overlap in time and compete for the disk. This causes slower performance as enforcing truly sequential accesses is much more difficult.

The other configuration uses two drives. In this case, the raw collection was kept on one drive and the temporary files were stored on another drive, eliminating any competition between reads and writes at assignment time. Similarly, the final file and the temporary files were stored on different drives; it is sufficient to put the final file on the first drive where the raw collection is to eliminate any competition at merging time.

Before discussing the experimental results, please note that we purposely do not repeat here all the numbers we got from the runs. These results are detailed in [Guðmundsson et al., 2012].

For all setups, at assignment time, we observed much overlapping between CPU computations and disk requests thanks to OS and device optimizations which keep the processor (usefully) busy while waiting for I/O completion. Merging also greatly benefits from the prefetching done by the OS: the few large files are brought into memory before the data gets processed, reducing I/O cost.

The experiments, however, show that the magnetic devices exhibit very poor performance as soon as they have to perform random I/Os—for example, the Intel SSD does I/Os for 46 seconds during the assignment phase while it requires above 500 seconds for the magnetic Seagate drive. This is not a surprise, moving their mechanical parts is time consuming. In contrast, SSDs do not suffer from random I/O patterns. SSDs are typically 3 to 10 times faster than magnetic disks. This explains the very small times, in particular with the Intel SSD which proves to handle competing reads and writes very well. Interestingly, the SuperTalent SSD performs poorly—unfortunately, not all SSDs are equal, as reported in [Bouganim et al., 2009]. In contrast, the Intel SSD handles random reads and writes very well.

By using separate physical drives for the reading and writing, competition for the disk is potentially eliminated. As the process typically does sequential and random I/Os, using an SSD for performing the random operations provides dramatic total time improvements. Using two SSDs drives is even more efficient, but the improvement is not as dramatic as when only random accesses go to SSDs.

One key lesson is that it is not necessary to put SSDs everywhere, which could be terribly expensive, but to use them solely where random accesses are massively needed. This greatly reduces costs, both in terms of performance and money.

IV.5. Concluding Chapter IV

In this chapter, we toured the high-dimensional indexing schemes that are built on top of vectorial quantization foundations. We came to working on such schemes after having extensively worked on scalar quantization schemes; the vectorial gain proved to much improve the performance of indexing schemes. Most of the work achieved here is more related to having acquired

a good understanding of the domain than it is to having designed a complete scheme, in contrast to what we did with the NV-Tree. Therefore, we met less database objectives here. We never seriously addressed the durability component, neither dynamicity; we directly addressed resource consumption with the balancing of clusters, as well as addressing both the CPU and the I/O bottlenecks with eCP.

Experiments with eCP proved it works very well. Yet, increasing even more the scale at which system should work requires to harness the power of multiple cores. Discussing this is the purpose of the next chapter.

Chapter V

Addressing Scale: Working with Millions, Billions, Terabytes

Contents

Multimedia collections have reached sizes that were unthinkable few years back, as this was highlighted in the introduction of this manuscript. A lot of papers published in multimedia research venues have experimental sections where the collections used for performance evaluations contain several millions of images or billions of descriptors. Simultaneously, the processing power of each computer has also grown. Architectures are now all 64bits, allowing for huge on-board RAM capacities, tens of gigabytes are not uncommon, and hundreds possible. All computers now come multi-cored and thus writing parallel programs is no longer reserved to some elite, equipped with exceptional machines, but has become the norm. At the same time, more hardware becomes available, thanks to easier access to Grids and/or Clouds.

Overall, such architectures are appealing when processing massive collections of multimedia material, especially when *creating* high-dimensional indices. Taking a raw collection of high-dimensional descriptors and creating for it an index to allow subsequent ultra-fast searches is still a long, complex, costly, and resource-consuming task. When the raw data collection is

on the order of terabytes of high-dimensional descriptors, as is the case when indexing tens of millions of real world images and directly using SIFT, then indexing may take days or even weeks.

Parallel and distributed architectures are also needed when *searching* indices in order to exhibit a very high query throughput. This is slightly different from the traditional case where an indexing system is used to rapidly return the points from the collection that are the most similar to the query point. In this case, a very small percentage of the indexed data is read from disks, one or few clusters at most. This, in turn, optimizes the response times of each query. Other applications need throughput, where sacrificing individual query response time is acceptable so that multiple queries can be run simultaneously in a batched mode. Real world applications concerned with very large image collections are likely to optimize for throughput—this is the case for Copyright infringement applications that typically need very high throughput to scrutinize such large amounts of multimedia material.

This chapter discusses a complete high-dimensional indexing approach harnessing as much processing power as possible from using parallel and/or distributed computing [Guðmundsson, 2013, Moise et al., 2013, Shestakov et al., 2013]. The approach we propose truly copes with very large scale datasets, typically hundred million images, dozen billion descriptors, tera-bytes of data. The parallel/distributed creation of the index as well as the parallel/distributed batch searching procedure both scale very well. The indexing scheme presented here is built on top of eCP, which has been described in the previous chapter. We first motivate the use of eCP in Section V.1. Then, we propose in Section V.2 a multi-threaded version of eCP to harness the power of the multiple cores we find nowadays in every machine. We go one step further and propose in Section V.3 a Map-Reduce based version of eCP that runs on top of the Hadoop distributed framework. Section V.4 concludes this chapter.

V.1. Addressing Scale with eCP

Although eCP was described in the previous chapter, we briefly recap here its fundamental principles.

eCP is a centralized high-dimensional indexing strategy. eCP is very related to the well-known k-means approach. As k-means, eCP adopts an unstructured vectorial quantization scheme to create clusters containing similar descriptors. eCP is designed to be I/O friendly as it assumes the data collection is too large to fit in memory and must reside on secondary storage.

eCP randomly picks C points from the collection that are used as the representatives of the C clusters the algorithm will eventually build. C is determined from having set the average number of data points each cluster should contain. This number is called the TargetSize, ts, and $C = N/ts$ where N is the number of points in the collection. eCP then organizes the C representatives in a multi-level hierarchy composed of L levels. The points from the data collection that remain are read one after the other, traverse the tree of representatives and are eventually assigned to the closest cluster representative at the bottom of the tree. The multi-level hierarchy allows to assign points with a logarithmic complexity. Once all the data collection has been processed, then eCP has created C clusters stored on disk as well as a tree of representatives, small enough to fit in main memory.

Searching with eCP requires to navigate down the tree of representatives by following the path indicated at each level by the representative that is the closest to the query point. Then, the corresponding bottom cluster is fetched, and the distances between the query point and all the points in that cluster are computed to get the k-nearest neighbors.

eCP compensates its somewhat brutal clustering by adopting ideas from various state-of-art indexing schemes. It can use a form of soft-assignment [Chierichetti et al., 2007, Philbin et al., 2008] while building the tree of representatives. With soft-assignment, each representative is not solely assigned to its closest parent representative, but it is assigned to its α closest representatives (typically 3). Note α applies only to the tree of representatives, not to the data stored in the clusters. It can also use a form of multi-probe approach at search time as more than one cluster can be searched, as it has been proposed for LSH [Lv et al., 2007, Joly and Buisson, 2008]. eCP can probe the β clusters that are the closest to the query point.

V.1.1. The case for eCP

We decided to augment eCP with parallel and distributed capabilities and not another state-of-the-art indexing solutions for the following reasons:

- eCP is quite representative of the core principles underpinning many of the unstructured quantization-based high-dimensional indexing algorithms that proved to perform very well and that make the bulk of the most recent high-dimensional indexing literature [Sivic and Zisserman, 2003, Jégou et al., 2011].
- eCP is not iterative by nature while other traditional indexing schemes based on k-means are. At every iteration of a k-means process, and in order to eventually converge, new representatives must be computed based on the previous round. When considering parallelism and/or distribution, a round of execution that is finishing must construct a global state of the computation, subsequently propagated to all processing participants, before iterating and initiating the next round. This is typically complex and costly, especially when no easy means for sharing information exist (e.g., no shared memory) as messages must be sent all over. Having no such rounds with eCP was a strong motivation for using this algorithm as distributing and parallelizing it were greatly simplified.
- eCP pre-calculates a representative hierarchy that is used to significantly speed-up the assignment of points to clusters. This is key for performance when data collections are terabytes sized in order to have a indexing approach usable *in practice*.
- eCP proved to return good quality result despite the crude process it uses to create clusters. We showed this earlier, but we provide more details on quality later in this chapter.
- Due to the extreme simplicity of its search procedure, eCP indeed covers a large spectrum of existing indexing approaches. A behavior very similar to the one of VideoGoogle ([Sivic and Zisserman, 2003]) can be obtained if instead of computing the distances to all the points in the fetched cluster(s) eCP simply returns the cluster identifier, as each cluster is indeed a visual word. It can also behave quite similarly to the vectorial variant of LSH when it processes the contents of the clusters [Paulevé et al., 2010]. eCP is also compatible with the best indexing solutions that are known today and that rely on some form of smart descriptor aggregation [Jégou et al., 2012]. Finally, eCP does not include any post-processing step to eliminate false positives. This can of course be added if needed.

V.1.2. What to Parallelize, What to Distribute?

Using an high-dimensional index for extremely large data collections requires distribution and parallelism. In this section we motivate the need for making parallel/distributed the creation of the index as well as the search phase in order to achieve high throughput.

Index Creation

The index creation process of eCP can be split into two main phases. During Phase #1, the creation of the index tree, cluster representatives are picked from the collection and organized in a in-memory tree. During Phase #2, vectors are assigned to clusters.

Obviously, Phase #2 is the prime candidate for parallelization and distribution. It is clear that chopping the entire data collection into independent parts assigned to physically distinct CPU/nodes is going to speed up the whole process.

In contrast, Phase #1 is in comparison computationally cheap. Therefore, picking C random points and building the in-memory hierarchy is best done on a single core once ts, α and L have been set. The resulting hierarchy of representatives is then made visible to the various processing units involved in the construction of the index. Each unit will use this hierarchy to assign its subset of the data collection to clusters. Results will be consistent across units as the hierarchy is identically replicated everywhere. Note that Phase #1 is executed only once, before launching the index creation process.

Searching

In contrast to indexing, it is hard to split the search process in phases. Basically, searching boils down to accessing one cluster and then finding the k-nn of a query point. In this case, a tiny portion of the overall indexed data is read from disks. Reads are typically random, because it is very unlikely that two consecutive query points will need the same cluster. Performance are here typically limited by this I/O bottleneck. No much CPU is used for finding the k-nn within the data read.

When local description schemes are used, then a series of query points are used, and k-nn are determined for each. Note that this is already a form of batching, but the batch typically consist of all the descriptors from a *single* image. In this case, many randoms reads are issued (typically few hundreds to few thousands), one for each query descriptor of the searched image. It is possible to reorder the I/O requests to enforce as much as possible sequential reads. It is thus necessary to determine which cluster is needed for which query point, before accessing any data on disk. Once this knowledge is acquired, then reordering the accesses is easy; in turn, the k-nn of the multiple query points for the searched image will be determined according to this order maximizing sequentiality, instead of according to the order with which they were extracted from the query image. There are however very few I/O requests for one image, compared to the overall number of existing clusters—few thousands cluster needed compared to few millions existing. Therefore, sequential access is unlikely, large jumps between I/Os are likely, overall making the I/Os still totally random.

Batching many query images at once potentially provides more dramatic performance improvements. Batching is interesting because it is possible to optimize the disk I/Os as everything that will be accessed is known in advance, allowing for careful resource planning. This is traditional in database systems. Batching is optimizing *throughput*. In contrast, searching the images that are similar to a single query image, as it is typically done in most systems, optimizes *response time*. Batch searches, as they serve many query images at once, sacrifice the response time of each individual image search for instead providing high throughput. Dealing with batches becomes interesting only when the batches are sufficiently large. Typically, specific profitable optimizations become possible once the batch reaches few thousands to few hundred thousands images, i.e., when it contains about 10^5–10^8 query descriptors. In this case, at least two optimizations can be made. They both make use of the list of the clusters identifiers that all the query points in the batch will eventually access.

Removing Duplicated reads. The first optimization removes duplicated reads as it eliminates redundant cluster requests. When the batch is sufficiently large, then it is possible that several different query points need to access the same cluster. With no special care, several I/Os could be issued to read this same cluster again and again, resulting in poor performance. In contrast, discovering that several query points need the same cluster allows to issue a single I/O. Once that cluster read, then several query points can be answered at once and their respective k-nn found.

Reordering I/Os. The second optimization has already been mentioned: it is the reordering of the I/Os. When the batch is sufficiently large, then it is likely the reordering will enforce a much more sequential reading pattern than a random one, jumps over the disk are likely to be smaller. Furthermore, it might be possible that two (or more) consecutive clusters are needed. In this case, the data prefetching by the operating system become fully beneficial. Overall, the cache hit ratio is improved.

Eventually, the batch can become so large that the search switches from being I/O bound to being CPU bound. It is possible that so many clusters are needed that it is worth enforcing a complete sequential read of all the clusters. In this case, high CPU cost is to pay for determining the respective k-nn of the many query descriptors that need every single cluster. At this point, multi-threading the search becomes a viable option.

We now move to describing the indexing and searching processes of eCP when running on multi-core machines for parallelism. Then, we will discuss distribution.

V.2. eCP on Multi-Core Machines

It is tempting to implement an index creation/search algorithm such that it can fully utilize the multitude of cores available with the current off-the-shelf hardware. Furthermore, vendors include in cores hyper-threading technology to improve the parallelization of computations. In addition, implementation is greatly simplified since the parallelism is obtained without the need to distribute computations; a simple thread library provides the basics for multi-threading on the many cores, and semaphores are sufficient to prevent simultaneous updates of shared data structures.

However, getting the most out of modern hardware is not at all straightforward. On the one hand, using too many cores gives less performance improvement because it is difficult to feed so many processing units with enough data to keep them as busy as possible. On the other hand, a lot of implementation details matter and standard monitoring tools poorly show what is going on at low level, little helping programmers to take the right design decisions for getting as much as they can from their machines.

This section describes the implementation of the index creation and search processes on a multi-core architecture. It also shows, for a variety of situations, where the bottlenecks come from and why they severely limit the performance. The section also provides a few rules of thumbs for programming high-dimensional indexing applications on multi-core machines. We however start with briefly reviewing hardware.

V.2.1. Background: CPU, Cores, Hyper-Threading, Caches

For a long time, the mainstream CPU development was focused on making CPUs faster and faster. Around the new millennium, CPU vendors observed they could not get over the 4GHz barrier without overheating problems. They instead reached more computing power by multiplying the number of processing units.

Multi-core CPUs were not new as they have been around for a long time, in specialized systems only. But around 2005, multi-core architectures hit the mass market and everyone could have a laptop/desktop with many cores, by default. Today, a standard desktop has 2–4 cores, and a single server can have 2 or even 4 CPUs, where each CPU has 4–24 cores.

To further improve the performance of the computations, the hyper-threading technology was introduced. For each processor core that is physically existing on the chip, the operating system sees and addresses two logical processors. It thus tries to share the workload between them when appropriate. The basic idea behind hyper-threading is to execute the other thread when the first one gets stalled due for example to a cache miss or some data dependency. By utilizing the otherwise idle CPU cycles, the performance is improved. Please note that the two logical processors that are seen by the operating system have different performance: two identical tasks ran on a single processor with hyper-threading will not complete at the same time, one is typically 30% slower than the other. The two logical units perform differently. It should be clear that, while all threads run on *logical* cores, some run on *real* cores while others run on *virtual* cores.

With modern CPUs, caching data is key to performance. Various levels of caches exist, typically in a hierarchy where small fast caches close to the processing units are backed up by larger but slower caches. The smallest (e.g., 64KB) and fastest cache is termed L1, and each core has an L1 cache. When the processor needs data, it first looks it up in this cache and the next larger cache, L2, is checked on a cache miss. When shared between the multiple cores of a single CPU chip, L2 caches are larger, 2–6MB; when not shared between cores, then they tend to be smaller, e.g., 256KB. The latest multi-core machines typically have an additional layer of cache, L3, that is either on-die shared or globally shared between all dies.

Performance of applications of course depends on the number of cores they use for running their computations, on the nature of the computation, as well as on their access pattern to data. High-performance computing is typically made of complicated iterative/recursive computations using little data as a starting point, using again and again the same data or transient values from one iteration to the other. Once the calculation starts from the data in memory, it proceeds and runs instructions on the data that is cached very close to the processor, requiring to access RAM every once in a while and when the calculation is over to save the final result. In this case, the CPUs are constantly busy as very few cache misses occur, and the underlying hardware is not stressed.

This is not at all the case for high-dimensional indexing and retrieval applications for which consuming data is central. All the work is data-driven, not calculus-driven. Data is fetched from disks, then installed in memory and then transferred to the processor where it does not stays that long before being moved back elsewhere. Once processed, data has to follow the inverse route at index creation time, as each data item is then typically assigned to a group of similar elements stored on disks and later used during retrieval. Data locality and the underlying hardware are therefore key for the performance of such *big data* applications.

V.2.2. Multi-Threaded Index Creation

The code that is doing the assignment of descriptors to representatives is very simple. It has been sketched earlier, see Chapter IV, Section IV.4.3, page 74. As the data collection is much larger than the RAM, it works in rounds where large chunks of raw descriptors are read from the disk, assigned to representatives, written back to disk. The costly part is of course the assignment to representative phase which can be multi-threaded very easily.

Before starting the creation, multiple threads are spawn and kept in a thread-pool, waiting for data. Then, a chunk of data is fetched into memory by one specific and dedicated thread. It makes other threads active and the contents of that chunk is divided evenly between these active threads. Each thread thus processes a small fraction of that chunk. The processing basically computes all the distances between the raw descriptors each thread has under scrutiny and the representatives in order to find the closest one. As each thread processes a disjoint portion of the chunk, no special synchronization is needed. The tree of representatives is accessed in read-only mode, also avoiding the need for any locking. Another dedicated thread pushes data on disk when assignments are complete.

The number of threads that are spawn is of course key to performance, as we will demonstrate later in the evaluation part. There is one particular value for that number that matters. When it is equal to the number of cores, then we are sure all threads will run on real hardware. When we go above this and run more threads than there are cores, then hyper-threading kicks-in and some threads run on real cores while others run on virtual cores.

V.2.3. Multi-Threaded Batch Searching

Batching assumes the system has hundreds or thousands query images to search simultaneously. Of course, image features are extracted from these images and build the batch to search. It is obvious that the features must be tightly linked to the identifiers of the images they have been extracted from.

As for the index creation, a pool of threads is created. One thread reads the batch of query descriptors as well as the tree of representatives. Then, each query descriptor goes down the tree in order to identify the cluster it will need, and that the search will later fetch to eventually determine its k-nn. At the end of this process, once all the query descriptors have traversed the tree, all the clusters that will have to be fetched from disk are known, but none have been accessed yet. It is therefore easy to create a lookup table allowing to quickly know what are the query descriptors that need a particular cluster.

That same thread wakes-up the other threads and then starts to fetch the clusters in order, to maximize the contiguity of the I/Os. Each fetched cluster is inserted in a producer/consumer queue. The producer is the thread fetching clusters. Consumers are the other threads. Each consumes a cluster from the queue and then scans it against the relevant query descriptors, known from the lookup table. Scan does the distance calculations which, in turn, updates their k-nn lists. When a thread is done scanning its current clusters, it goes to the queue to consume another one, if any.

Once all the clusters that were relevant for the current batch have been scanned, then vote aggregation is done to get the final results. This possibly loops to process the next batch.

V.2.4. Experimental Setup

We now report some experiments designed to facilitate detailed performance analysis of the multi-threaded versions of the index creation and the batch search.

All the experiments reported in this section were run[1] on a Dell r710 machine that has two Intel X5650 2.67Ghz CPUs. Each CPU has 12MB of L3 cache that is shared by the 6 real and 6 virtual cores. There are therefore 24 logical cores, 12 being real processing units. The RAM consists of 18x8GB 800Mhz RDIMMs chips for a total of 144GB. For secondary storage, this machine uses a Dell PowerVault MD 1200 with 12 15k-rpm, 600GB SAS disks organized in a single RAID6 configuration by a Dell PERC H800 controller for a total of 5.7TB usable disk space. Experiments were in part ran while no other user was running concurrently.

We indexed and then searched two image collections for these experiments (see II.4). Briefly recalling them here, the first collection has 100,000 random Flickr pictures resulting in more than 110 million SIFT vectors, for a total of almost 13.6GB of data. The second collection contains 30 millions 150x150 images from Flickr, resulting in 8 billions SIFT descriptors for a total of almost 1Tera-Bytes of data. The indexed collections contains the Copydays and the 49k ground truth, allowing to easily obtain recall and precision results. Batches at querying time vary in contents and sizes, as described later.

We first present the performance for the multi-threaded indexing process. We then move to the multi-threaded batch search process.

V.2.5. Performance of Multi-Threaded Indexing

The first experiment determines the performance gains when using an increasing number of cores. For this experiment, we used the 110 million SIFT collection that is indexed by assigning the descriptors to 111,424 representatives organized in a 3-levels hierarchy. Experiments are split in two parts. The first part uses only up to 12 cores, and the machine settings were in this case such that the use of hyper-threading was disabled; only real cores could possibly be used. The second part enables virtual cores and hyper-threading, thus utilizes up to 24 logical cores.

Real Cores

We started by running the index creation using only two cores in order to have a baseline. With two cores, it takes approximately 6,300 seconds to index the descriptor collection. We then ran the same index creation again and again, varying the number of cores, however, from 2 to 12. We normalized the response times with respect to the baseline and report in the Figure V.1 the resulting relative response time reductions when varying the number of cores used during index creation.

On this figure, the large green area bounds the best possible theoretical response time reduction: as the normalized response time is set to 1 when using two cores, then it could only be 0.5 when using 4 cores, 0.25 when using 8 cores, etc.

The red line called “L2” shows the relative response time observed during the index creation when increasing the number of cores. When using 6 cores, the response time is only 0.37 times the one observed when using two cores, while the best in theory is 0.33. With 12 real cores, 0.20 is observed while theory would give 0.17. As expected, the response time is significantly reduced when using more core, yet, some overhead starts to be visible. Switching contexts, cache competition and RAM latency tend to make each thread “less efficient”.

[1] This is the same experimental environment as the one used Section III.4.7, page 42.

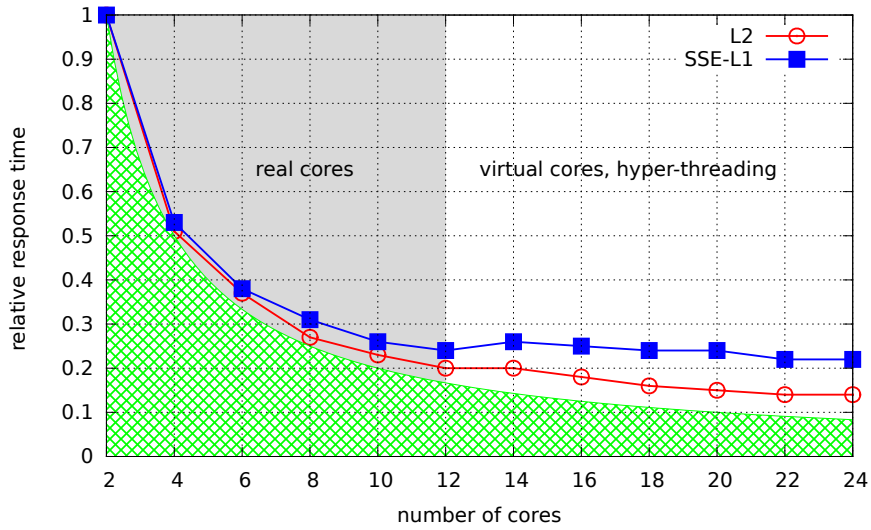


Figure V.1.: Relative response time, varying the number of cores. L_2 and SSE-L_1

Real and Virtual Cores, with Hyper-Threading

We then changed the settings of the machine, allowing hyper-threading. In this case, hyper-threading on virtual cores kicks in as soon as more than 12 cores are used. The response time continues to decrease but the gain is less significant. For example, the response time with 24 cores is 0.14 times the one observed when using two cores while 0.08 could be theoretically expected. It is however key to understand that the additional cores are virtual, and thus it is unrealistic to assume they could absorb work as real cores do. Intel specifies that each virtual core with hyper-threading improves the performance by roughly 20–30%. It is therefore normal to see the curve increasing its gap with respect to the theoretical trend line once we hit hyper-threading: assigning to clusters the descriptors stored in some of the fractions of the raw data chunks takes more time.

Note, however, that our implementation of the multi-threaded index creation emphasizes this phenomenon. We split the work evenly between all threads, creating as many fractions as there are threads to run indexing. Each thread has therefore quite a lot of things to do, and the difference in their speed becomes visible. In contrast, if the work is divided into tiny fractions, creating much more small fractions then there are possible threads, then the overall behavior improves. Threads are much shorter, and they pick another fraction to process on a real core as soon as it is possible. Hyper-Threading is still slower on virtual cores, but the observed performance in this case are getting very close to the theoretical ones, augmented by 30% once hyper-threading kicks-in (this is not shown on the figure).

Note that the operating system of the machine does not make any distinction between real and virtual cores. Therefore, tools monitoring the performance of machines (such as the well known `top` or `time` Unix commands) return wrong information when dealing with multi-core architectures. Those tools for example considered that our machine had 24 real cores, and used that information to compute load information. In reality, with hyper-threading, the machine is much more loaded than displayed, as virtual cores are not as efficient as real cores.

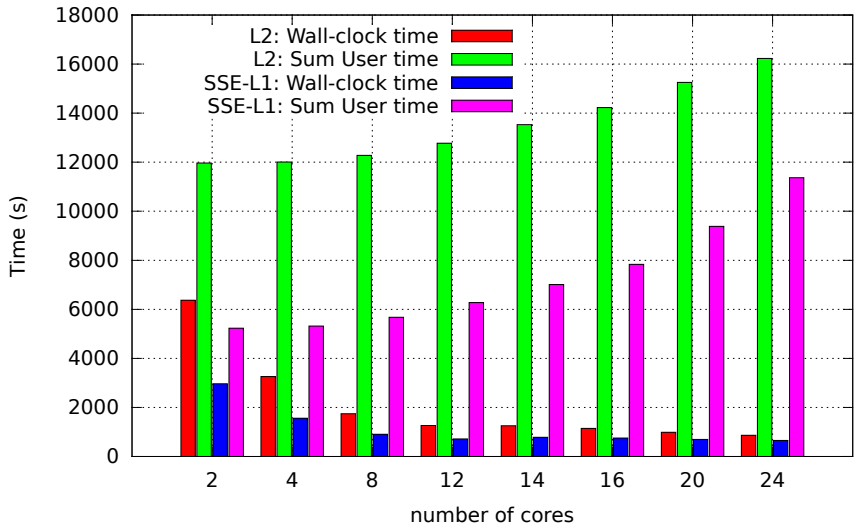


Figure V.2.: Response times, varying the number of cores. L_2 and SSE-L_1

Streaming SIMD Extensions–SSE

Most machines come with sets of hard-wired instructions, allowing for even more performance improvements. These instructions, referred to as SSE, are part of a SIMD instructions set where a single instruction is applied on multiple data elements during one single processing cycle. This is typically useful for computing distances where the same basic mathematical operation is applied to the components of two high-dimensional vectors.

The Intel machine we are using has SSE instructions for computing L_1 distances. While SIFT traditionally needs L_2 distances, we decided to create an index using the L_1 SSE instructions to observe the performance improvement. We are well aware, however, that this is no good quality wise: the "natural" proximity of vectors is not preserved, giving in turn poor search results. Nevertheless, that experiment is interesting, response time improvement wise.

Figure V.1 shows the performance improvement relative to the baseline using two cores (the baseline is determined from the time it takes to run the creation using L_1 SSE instructions on two cores), reported by the blue line named "SSE-L1". It is interesting to see the improvements are less pronounced compared to the L_2 case. For example, with 12 (real) cores, the response time is 0.24 while the theoretical is 0.17 and 0.20 were observed when no SSE instructions were used. With 24 cores, we observed 0.22 while 0.08 could be expected; 0.14 were observed for L_2 distances.

Overall, SSE instructions increase the pressure on the cache and on the access to the RAM as the CPU must be fed with data more frequently, instructions processing data more rapidly. This experiment shows the machine suffers from cache misses. The CPU is too fast, and resolving cache misses kills performance. We give a deeper view on this phenomenon in a coming paragraph.

Response Times

Figure V.2 reports the observed performance when indexing the 110 million SIFT collection, for the cases where the index creation uses the natural distance for SIFT (L_2) as well as the SSE-L_1 distance. This figure gives the time the index creation takes (the red and blue bars

# of levels	cluster size	# of representatives	Tree Size (MB)	Indexing Time (h)	IPC_{100} L_2	IPC_{100} L_1
2	500K	16,245	2.2	-	1.25	0.61
3	50K	162,446	21	33.05	0.88	0.28
4	5K	1,624,452	220	40.06	0.73	0.19
5	1K	8,122,260	1,100	42.71	0.68	0.20

Table V.1.: Characteristics of the index tree, and performance, varying # of levels

give the wall clock times for L_2 and L_1, respectively) as well as the sum of the user times by each thread when varying the number of cores (green and pink bars). Two comments are in order. First, running the application with SSE instructions is much faster. When real cores are used, it is roughly two times faster. Once virtual cores and hyper-threading get fired, then the improvement slows down. Using SSE with 24 cores is only 1.28 faster compared to non SSE L_2 instructions. Second, the green and pink bars giving the sum of the times it takes for each thread to process its own data get bigger and bigger as the number of cores grows. The index creation finishes more quickly thanks to the increased number of cores, each core has to do more work because of cache and memory competition: each thread is idle more often, resulting in a faster but less profitable computation.

Watching for Cache Misses

From the experiments described above, the latency of the RAM has the most impact on performance. We therefore created a series of high-dimensional indices that differ by the size of the multi-level tree indexing the data, which, in turn, changes the cache hit ratio. We are only concerned with performance here, not with quality of searches. We used 12 cores and disabled hyper-threading. We used the largest collection composed of 8 billion SIFT descriptors to be in a realistic setting, and created 4 different indices having respectively 2, 3, 4 and 5 levels by changing the value for the desired cluster size. This, in turn, requires to use a different number of representatives: shallower trees use few representatives, thus consume less main memory, in contrast to deeper trees. The characteristics of these different indices is summarized in the Table V.1. The shallower index tree is only 2.2MB in RAM, which means it can entirely fit in the L3 CPU caches (that are 12MB). As this tree is used to determine the representative that is the closest to each point of the data collection, having it in the cache is good for performance.

We then launched the creation of the index using only 4 cores, no hyper-threading. The runs were aborted after having processed about 40GB of data—the goal was not to run to completion, but rather to snapshot the behavior of the system after having processed enough data to make sure stability and cruise speed is reached.

We measured the number of instructions that are executed for every 100 CPU-cycles. The more instructions, the more "actually productive" is a CPU; in contrast, a low number of instructions per 100 cycles (IPC_{100}) shows the CPU is mostly idle, typically waiting for data to process. That number of instructions is also reported in Table V.1. It is possible to observe that, overall, IPC_{100} is globally small. This is to contrast with typical high-performance applications mostly doing calculations. In this case, little data is needed in comparison to the what is needed for index creation. Memory latency is causing the CPU to remain idle most of the time, as it can not be fed with data quickly enough. This is corroborated by two series of figures reported in this table. First, IPC_{100} is increasing as the size of the index tree decreases—the cache hit ratio

grows, increasing the profitability of each CPU cycle. Second, IPC_{100} are much smaller when using the SSE instructions: this is normal as these instructions are much quicker to run, and, in turn, cache misses are occurring at a faster rate. It is also interesting to report that memory `load` and `store` operations represent roughly 50% of the time it takes to create indices with the SSE-L_1 instructions, while this represents roughly 35% when using the natural SIFT distance. This shows again the increased demand on the memory cells.

Multi-Threaded Indexing: Discussion

Overall, the experiments reported here, as well as other experiments, not reported, suggest to take care of the following details when implementing a code that is as resource consuming as is the creation of one high-dimensional index on a multi-core architecture:

- It is absolutely essential to buy RAM that has low latency—the access patterns to data during index construction is extremely random, therefore, data is very often needed from the RAM as it is not found in any of the caches close to the CPU. The experiments reported here essentially show the effects of RAM latency;
- We implemented several ways of spawning threads. The most efficient one is to rely on a pool of threads created once for all, each processing a small segment of the data collection. This is in contrast with creating new threads on-demand and/or having each thread processing a large amount of data. Processing relatively small data segments (i.e. 128MB) is best because the application does not have to wait a too long time for the slowest thread to finish before completing. With small data segments, each threads finishes quickly and then loops to process the next pending data segment;
- SSE instructions should be used when possible, as they really boost performance. They do put more pressure on RAM accesses, further necessitating low latency memory chips;
- Hyper-threading helps with this type of application but not that much, again because accessing data is mostly random, and utilizing idle CPU cycles is often difficult because data is not at all there as it can not be brought fast enough to the CPU;
- From a practical point of view, traditional Unix tools for performance measurements are not very helpful when measuring multi-core applications. We recently discovered `tiptop`, a monitoring tool designed for parallel machines. It is built on the Unix `top` command, and it gives to non-expert users few simple low level metrics helping taking the right design decisions when implementing on multi-core architectures [Rohou, 2011].

V.2.6. Performance of Multi-Threaded Batch Searching

This series of experiments intends to show the performance of multi-threaded batch searching process. We use the large indexed collection that was described previously, i.e., 8.1B SIFT descriptors from 25M images. As before, and to show the impact of the desired cluster size, we run queries against that collection clustered into 8.1M clusters as well as into 1.6M clusters. It is important to recall here that these collections contain the Copydays and the 49k ground truth, allowing easy and objective benchmarking. Since we check quality from a quasi-copy perspective, consistently with all other quality experiments in this manuscript, we will give figures for Recall at 1.

The first experiment discusses quality. We then move to discussing throughput.

cluster size	# of levels	# of representatives	Copydays Recall at 1		49k Recall at 1	
			probe 1	probe 3	probe 1	probe 3
5K	4	16,245	0.8668	0.9011	0.9503	0.9657
1K	5	8,122,260	0.8448	0.8733	0.9358	0.9656

Table V.2.: Quality of batch searches, varying desired cluster size (5k and 1k). $k = 20$. 25M images, 8.1B descriptors, 1To

Quality of Batch Searches

Table V.2 gives the quality of the search when indexing about 25M images according to two index configurations. The Recall at 1 is given for two strategies enforced at query time. The first strategy identifies the closest cluster for each query point, fetches it and then computes k-nn. This corresponds to the "probe 1" columns in the table. The other strategy corresponds to the multiprobe case, already discussed elsewhere in this manuscript. In this case, the three closest clusters are fetched for each query point, and the k-nn are then determined from all the descriptors from these three clusters. This corresponds to the "probe 3" columns in the table. Obviously, probing multiple clusters increases the quality of the searches, since we are more likely to find the correct neighbors inside this extended set of points. Processing a larger set of points also explains why the quality results are better when clusters typically contain on average 5,000 points compared to when they contain about 1,000 points only.

Overall, the search quality is extremely good. We know Recall can not be equal to 1.0 because there are some query images with very few (a handful) descriptors (sometimes none!). We typically identify the correct k-nn of less than 300 query descriptors lost in the middle of 8.1 billion others... This is another indication showing eCP is a very viable technique.

We now turn to discussing throughput.

Throughput of Batch Searches

The motivations for running batches of queries were given above. We expect to observe here improved performance for two reasons. First, identifying all the clusters that are needed before accessing them allows to eliminate redundant reads, saving I/Os. Second, a large batch is likely to increase the number of clusters to fetch from disk, which in turn increases the contiguity of accesses and makes prefetching more profitable. Therefore, it is key to monitor (i) the number of clusters requested per batch as well as (ii) the number of redundant accesses eliminated. We will however start by discussing the nature of the batches before moving to the throughput performance of batch searches.

Nature of Batches. The Copydays and the 49k ground truth and query sets have been built by creating several variants of images. Therefore, by construction, there is a certain amount of overlap between the variants of a single ground truth image—these variants are somehow similar one another. As a batch contains all these variants, then the processing will benefit from this overlap: fewer clusters are concerned, many query points from the variants indeed need to scan the same clusters, etc. To some extent, this might not reflect what a typical batch can contain, where the payload is possibly completely random, as if the batched query images were submitted by many different users through a Web service. In this case, there is no reason to have similar

Batch Size (# images)	Batch Name	# of Query Descriptors	# Cluster Requested	% of the DB Fetched	Passive Q-desc
3,055	Copydays	955K	420K	25.86%	56.00%
	49k	940K	363K	22.33%	61.43%
	Random	967K	585K	36.02%	39.48%
48,883	49k	15.0M	1.4M	86.00%	90.71%
	Random	15.6M	1.5M	92.97%	90.32%
100,000	Random	31.2M	1.6M	97.06%	94.94%

Table V.3.: Nature of batches. 1.6M index. Probe 1

Batch Size (# images)	Batch Name	# of Query Descriptors	# Cluster Requested	% of the DB Fetched	Passive Q-desc
3,055	Copydays	955K	571K	7.03%	40.20%
	49k	940K	500K	6.16%	46.83%
	Random	967K	821K	10.11%	15.05%
48,883	49k	15.0M	4.1M	51.04%	72.43%
	Random	15.6M	5.2M	64.03%	66.67%
100,000	Random	31.2M	6.4M	78.86%	79.45%

Table V.4.: Nature of batches. 8.1M index. Probe 1

query images, and therefore, the degree of overlapping might be smaller. To study this effect, we constructed another batch of 100,000 queries, totally random, and that have no counterpart in the database. We are not interested by the quality of the results obtained when using this set, but rather only in the underlying performance of the search process.

We give in Tables V.3 and V.4 information on the nature of the batches. The collection that is indexed contains 25M images, 8.1B SIFT, 1Tb. That collection is used to create two databases, differing by their cluster sizes. Table V.3 displays information on the clusters various batches need when the index has 1.6M representatives, that is, when there are roughly 5,000 descriptors per cluster. Table V.4 displays information corresponding to the case where the index has 8.1M representatives, that is, when there are roughly 1,000 descriptors per cluster. These two tables are identical in their structures, presented now.

The first column gives the number of images that are in the batches used here. These sizes are aligned with the benchmarks we use here, to facilitate the understanding. For example, we ran a batch containing all the Copydays queries, that is, 3,055 images, as well as another containing the first 3,055 images from the 49k query set. The same applies to the batch containing 100,000 unrelated images. We therefore used three families of batches with 3,055 images (full size of Copydays), 48,883 images (full size of 49k) and 100,000 images. We then display in the third column the corresponding number of descriptors. It is rather stable across batches and increases quasi-linearly with batch size. The fourth column gives the number of different clusters that are requested by all the descriptors in a batch—redundant cluster requests have thus been eliminated from this count. We clearly see here key differences between batches.

One noticeable difference is the number of different clusters that must be fetched for these batches, respectively. The Random query batch needs to read 585k clusters while the two other batches need significantly fewer clusters. This is due to their nature, highlighted above. This

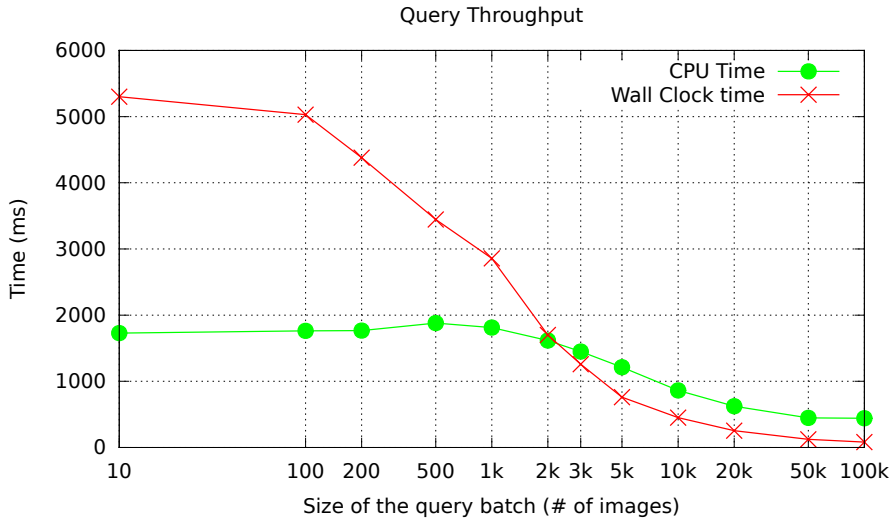


Figure V.3.: Throughput, random batch, varying size

difference is even more visible when considering the 8.1M index. Here, there are much more clusters, and the queries for the Copydays and 49k batches are clearly "concentrated" into a reduced number of clusters. The percentage of the database that must be fetched to answer the batches varies accordingly. Note that for large batches, almost all the clusters are needed–this is a clear indication that switching the disk access mode to sequential is indeed valuable, justifying the batched approach where I/O requests are reordered. This is particularly true for the 1.6M index, as almost every single cluster has to be fetched and scanned against at least one query descriptor. Overall, the likelihood of benefiting from prefetching increases as the size of batches grows.

The last column gives the number of "passive" query descriptor overlap, i.e., an estimate of the number of descriptors in a batch that issue a request for a cluster already issued by another query descriptor. It is computed by evaluating:

$$\frac{\#\ query\ descriptors - \#cluster\ requested}{\#\ query\ descriptors} \tag{V.1}$$

With the 1.6M index, about half of the query descriptors request a cluster for the 49k set while more than 85% of them do so for the Random query set. This clearly illustrates the differences in the nature of these sets, where much more (visual) redundancy exists for the 49k batch. Similar evidences apply to Copydays, which is less redundant than the 49k set, however.

Throughput. We have ran experiments collecting the time it takes to fully process batches. The intuition is that small batches (few images) should be completely I/O bound due to the total randomness of disk accesses, while larger batches should eventually be CPU bound. In this later case, accessing sequentially the disk is not a problem, but the large number of distance computations to perform on the CPU for each cluster dominates.

To confirm this intuition, we truncated the Copydays, the 49k and the Random query batches to contain only 10, 100, 200, 500, 1k, etc. query images. We ran these batches on 12 real cores, disabling hyper-threading, and instrumenting the search process and checking the logs. We solely discuss here the results obtained when using the Random batch as it is the more expansive to process.

When the number of images in the Random batch is very small, then the process is totally I/O bound. This can be seen on the far left of the Figure V.3. The wall-clock time it takes to process one image is very high while the CPU time it takes to determine its k-nn is small. This is due to the time wasted waiting for the data to arrive from disk for being processed. The machine issues a random I/O to get a cluster, and then is idle for a while. Once there, it takes little time to get the k-nn.

As we move toward the right of this figure, then two phenomenon can be observed. First, the idle time waiting for I/O completion decreases—this is normal, the access pattern becomes more and more sequential with better and better contiguity. Prefetching and caching become very beneficial. At one point, when the size of the batch reaches approximately 2,000 images (2k, x-axis), then the I/O bottleneck vanishes. The machine never waits for data to be brought-in from disks, the data is there already, cached. Reordering cluster requests creates more and more sequential I/Os as batches enlarge. Prefetching becomes more and more profitable. The next clusters to scan are here, loaded in RAM in advance. On the other hand, the CPU remains stable until it reaches about 2 to 3k images in a batch. At this point, the CPU per image decreases as well: several cores process the clusters that are in main memory, the system is less slowed down by the randomness of the access to RAM, and a better behavior of the caches can be observed.

We also made experiments when the amount of work for the CPU increases, by setting the number of clusters each query point is probing to 3. In this case, the CPU times go up again as the system becomes CPU bounded due to the very large number of computations to achieve. Note the use of the virtual cores when the load is large enough exhibits some slowdown, due to their intrinsically and normal lower power.

Overall, the time it takes to process one image from the batch drops from 5.3 seconds in the case of a batch with 10 images to about 82 milliseconds with the full 100,000 images Random batch. Rather identical results apply for the Copydays and the 49k cases.

V.3. Distributing eCP—DeCP

We just showed that the parallelized version of eCP can quite well make use of all the processing power of a multi-core machine. We indexed an image collection containing 25M pictures. That collection creates 8.1B SIFT descriptors and occupies about 1 Terabyte of data on disks. Indexing such a large collection, even when using a powerful machine having 24 logical cores and 144GB of RAM takes about 40 hours. Indexing larger collections with a single machine will thus be problematic.

Addressing even larger datasets requires a distributed approach. There is on the one hand the obvious need for more computing power. It is a better option to use many off-the-shelf machines rather than a single box with even more cores. There is on the other hand the need for better I/O management, and using many machines each having local storage is a good option too. In this case, disk parallelism is easy to get, and it is likely to outperform any complicated high-end RAID solution.

Distributing computations is not a trivial task. There are obvious problems related to synchronizing the various processes running on independent machines, there are failure problems, possible communication bottlenecks, scheduling and balancing the load, overall, all this is quite complicated. Adding these features to eCP would be a Herculean task.

Several frameworks have been recently proposed to ease this programming, such as Dryad [Isard et al., 2007], GraphLab [Low et al., 2012] or Map-Reduce [Dean and Ghemawat, 2008]. Such frameworks (almost completely) transparently handle these complicated issues, leaving

programmers to solely focus on their tasks, not on the plumbing required by distribution. Yet, frameworks facilitating the programming of distributed and parallel tasks on a grid impose on programmers very strict constraints sometimes conflicting with the properties of their applications. For example, Map-Reduce imposes a flow of data that makes iteration-based or graph-based search algorithms hard and costly to implement.

It is therefore not trivial to port any of the state of the art high-dimensional indexing technique to a grid environment. It often requires to change the design of the indexing algorithm itself to fit in the mold as well as to finely understand and then tune the many parameters of the framework facilitating distribution.

We describe here how the Map-Reduce paradigm can be applied to the eCP indexing algorithm, resulting in the DeCP (Distributed eCP) indexing scheme. We demonstrate that great scalability can be achieved using Hadoop, a popular Map-Reduce-based framework. We start however by giving a brief overview of the Map-Reduce paradigm, Hadoop and HDFS. We then detail the DeCP index creation with Map-Reduce before describing the Map-Reduce batch searching version of DeCP. We evaluate index creation and search using our image collection containing roughly 100 million images, this is about 30 billion SIFT descriptors or about 4 terabytes of data. The extensive experiments ran on this very large-scale dataset provide a basis for a discussion on the problems raised when managing so much data with Hadoop and a grid. We thus draw several lessons we want to share.

V.3.1. Background: Map-Reduce, Hadoop and HDFS

Map-Reduce

The Map-Reduce framework is originally by Google [Dean and Ghemawat, 2008] and it is a programming model for processing extremely large datasets. It exploits data independence to do automatic distributed parallelism. The developer is tasked with implementing the *Map* and the *Reduce* functions. The input data is distributed in blocks to the participating machines using the distributed Google file system GFS [Ghemawat et al., 2003].

When a job is launched, the system automatically spawns as many Map functions as there are data blocks to process, such that each spawned mapper is processing data where it resides, or as close to it as possible. Each mapper reads the data iteratively as a key/value pair record, processes it and, if necessary, outputs a key/value pair bound for a Reduce function. All records with the same key go to the same Reduce task. The framework thus includes a copy-merge-sort data shuffle step, where data from several mappers gets directed to specific reducers depending on their key. Once enough data is locally available to reducers, on disk or in memory, they process the records and produce the final output.

The Map-Reduce run-time environment transparently handles the partitioning of the input data, schedules the execution of tasks across the machines and manages the communications between processing nodes when sending/receiving the records to process. The run-time environment also deals with node failures and restarts aborted tasks on nodes, possibly on replicated data in case of unavailability. The framework uses as little network bandwidth as possible by processing data where it resides or at the nearest available node, paying attention to the network topology and minimizing reading over machine-rack boundaries.

Hadoop and HDFS

The Map-Reduce programming model has been implemented by the open-source community through the Hadoop project.[2] Maintained by the Apache Foundation and supported by Yahoo!, Hadoop has rapidly gained popularity in the area of distributed data-intensive computing. The core of Hadoop consists of the Map-Reduce implementation and the Hadoop Distributed File System (HDFS). Hadoop is now the de facto reference Map-Reduce implementation.

The architecture of Hadoop consists of a single master *jobtracker* and multiple slave *tasktrackers*. The jobtracker's main role is to act as the task *scheduler* of the system, by assigning work to the tasktrackers. Each tasktracker has of a number of available *slots* for running tasks. Every active map or reduce task takes up one slot, thus a tasktracker usually executes several tasks simultaneously.

When dispatching map tasks to tasktrackers, the jobtracker strives at keeping the computation as close to the data as possible. This technique is enabled by the data-layout information previously acquired by the jobtracker. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes closer to the data (belonging to the same network rack). The jobtracker first schedules map tasks, as the reducers must wait for the map execution to generate the intermediate data. Apart from data splitting and scheduling responsibilities, the jobtracker is also in charge of monitoring tasks and dealing with failures.

HDFS [Shvachko et al., 2010] was built with the purpose of providing storage for *huge files* with *streaming data access patterns*, while running on clusters of *commodity hardware*. HDFS implements concepts commonly used by distributed file systems: data is organized into files and directories, a file is split into fixed-size blocks that are distributed across the cluster nodes. The blocks are called *chunks* and are usually of 64 MB in size (this parameter specifying the chunk size is configurable).

The architecture of HDFS consists of several *datanodes* storing the data chunks and a centralized *namenode* responsible for keeping the file metadata and the chunk location. HDFS handles failures through chunk-level replication (default 3 replicas). When distributing the replicas to the datanodes, HDFS employs a rack-aware policy: the first replica is stored on a datanode in the same rack, and the second replica is shipped to a datanode belonging to a different rack (randomly chosen).

In addition to being used in cluster computing, Hadoop is becoming a standard for cloud computing. The generic nature of clouds allows resources to be purchased on-demand, especially to augment local resources for specific large or time-critical tasks. Several organizations offer cloud computing cycles that can be accessed via Hadoop. Amazon's Elastic Compute Cloud contains tens of thousands of virtual machines, and supports Hadoop with minimal effort.

V.3.2. DeCP Indexing with Map-Reduce

We describe here the Map-Reduce based implementation of the DeCP high-dimensional index creation scheme enabling the fast indexing of billions of descriptors, terabytes of data. This index creation technique scales very well with growth of the data collection and/or available hardware resources.

Indexing with DeCP boils down to adapting eCP to Map-Reduce. It is quite straightforward. Map tasks do the assignments. Each mapper loads the representative hierarchy and clusters the data by reading-assigning-emitting every descriptor in it's block of data. The key emitted is the identifier of the cluster the descriptor is assigned to.

[2] `http://hadoop.apache.org/`

Reduce tasks receive records grouped and sorted on their cluster identifier from the shuffle. All reducers do is to propagate to disk the data they receive to form the bottom level of the index, i.e., the clusters themselves. Note some bookkeeping is needed to keep track of cardinalities, etc.

V.3.3. DeCP Batch Searching with Map-Reduce

Consistently with the discussions above, we propose a Map-Reduce search scheme for DeCP that is geared toward throughput as it processes very efficiently large batches of queries.

Being a batch processing framework, Map-Reduce is not designed for answering individual queries, but is well suited for processing massive batches of queries. As mentioned previously, batching queries first requires to build a lookup table. Once built, it is easy to determine from the table which query descriptors in the batch must be scanned against a particular cluster given its id. Once that table is created, it is sent to all participating machines. Hadoop is then started.

All mappers receive the whole query batch with query descriptors ordered by cluster identifiers, and start to process their blocks of data. All blocks are read, but their contents is processed only if points they contain belong to the clusters needed by at least one query descriptor from the batch. Batching consumes RAM at each mapper since they have to maintain several k-nn tables for all the query points concerned with the current cluster under analysis. Tables can be deallocated when map tasks cross cluster boundaries, and a series of records are emitted. Special care must be taken when a single cluster spans more than one data block. Reducers finalize k-nn result lists for all the query points.

Overall, the cost for processing a batch is either entirely dominated by the cost of reading all data blocks, or dominated by the CPU for distance computations if the batch is really large or if there are very many points in each cluster.

V.3.4. Experimental Setup

We evaluate index creation and search using our image collection containing roughly 100 million images, this is about 30 billion SIFT descriptors or about 4 terabytes of data.

Distributed Environment on Grid'5000

The experiments were carried out on the Grid'5000 grid testbed [Bolze et al., 2006]. The Grid'5000 project is a widely-distributed infrastructure devoted to providing an experimental platform for the research community. The platform is spread over ten geographical sites located mostly through the French territory. We could get access to the machines belonging to the Rennes site. Grid'5000 is an exceptional high-end platform, a beautiful tool for research.

We must highlight that performance evaluation with Grid'5000 was sometimes problematic. Grid'5000 is a reservation-based system, and we could run tests only when it was our turn to use this intensively used machinery. Sometimes, our experiments failed, due to problems in the platform or to bugs or bad settings from us. As a consequence, we could not run all the desired experiments to nicely check the impact of running with various datasets, query batch sizes, block size configurations, mapper and reducer settings, varying the number of nodes, the degree of replication, etc. We can not discuss here a completely consistent view on the performance. This is unfortunate. The results we have still give us quite clear trends facilitating the understanding of what is at stake when distributing with Hadoop.

Cluster id	#Nodes	#CPU@Freq	#Cores/CPU	RAM	LocalDisk
Cl_1	64	2 Intel@2.50GHz	4	32GB	138GB
Cl_2	25	2 Intel@2.93GHz	4	24GB	433GB
Cl_3	40	2 AMD@1.70GHz	12	48GB	232GB

Table V.5.: Cluster configurations

%age	# of Images	# of Descr.	Data Size	# of Representatives	# of Levels	Tree Size
10%	10M	3.3×10^9	0.5TB	652K	4	193MB
20%	20M	7.8×10^9	1.0TB	1.5M	4	461MB
100%	100M	30.2×10^9	4TB	6M	5	1.8GB

Table V.6.: Index configurations

We could have access to 129 nodes belonging to our local grid infrastructure. The nodes form three clusters, each composed of identical machines, as it is reported in Table V.5. While each cluster has a highly connective internal network, inter-cluster bandwidth is limited. In practice, some of the 129 nodes may be down at any given point of time. The Hadoop framework was deployed as follows: the namenode, jobtracker and the job client are each on a dedicated machine, while the other nodes serve as both datanodes and tasktrackers.

At the level of HDFS, we use the default replication factor of 3 for the input data. In addition to facilitating the tolerance of faults, data replication favors local execution of mappers and minimizes the number of remote map executions, this being key for performance. We however typically set the output replication factor to 1 only. A larger value adds a substantial overhead to the running time because one replica goes to a remote rack. That cost becomes significant given the size of our data sets. We also set the chunk size at the level of HDFS to be 512MB (and not 64MB, default value).

Datasets, Queries and Ground Truth

The dataset we used in our experiments is the one created within the context of the Quaero project. It has been described Section II.4. It contains roughly 100 million images from the Web. Images have been resized to only 150 pixels on their largest side. SIFT descriptors were then extracted from these images, resulting in about 30 billion descriptors, i.e. 300 SIFT descriptors per image on average. To best of our knowledge, this image collection is one of the largest collections encountered in the content-based retrieval literature.

To evaluate the quality of indexing, we used that data collection as a distracting dataset into which we have drown the Copydays and the 49k ground truth. As before, the corresponding variants are used as queries, and, to assess quality, we then simply count how frequently the original images are returned as the top result. Many query images are visually such that only a very small number of SIFT descriptors can be extracted from their contents, e.g., 1% of the images have less than 8 descriptors. Finding the original images from their modified versions is therefore sometimes very challenging. Getting 100% accuracy is impossible as some image variants have zero SIFT descriptors (too dark e.g.).

To facilitate the experiments as well as to get a better understanding of the scalability issues, we indexed images subsets containing roughly 10% and 20% of the entire collection in addition

to indexing the full 100M images. Details on the resulting configurations are reported in the Table V.6. Clusters contain 5,000 points, occupying 645KB, on average. On the one hand this creates quite a lot of clusters, on the other hand, each is quick to analyze at searching time.

V.3.5. Implementation Details

Several implementation details must be clarified to know how DeCP works with Map-Reduce and Hadoop.

Preparing the Dataset. The descriptors extracted from the image set are stored as binary files comprising records of 132 bytes; each record defines a descriptor and consists of a 4-byte integer for the image identifier followed by the 128 bytes of the actual descriptor. Overall, the 30 billion descriptors occupy just below 4 TeraBytes on disks. We first implemented a conversion mechanism creating *SequenceFiles* from binary data. A SequenceFile is a Hadoop-specific data file employed for dealing with binary data. It consists of a header and one or multiple records. The header contains metadata that HDFS uses to parse the records. The records in a SequenceFile are fixed-sized and are defined as a single key-value pair. Several features (such as support for block compression and *sync markers* allowing to seek to the boundary of a record) make SequenceFiles a good choice for processing binary data with Hadoop. The descriptor conversion to SequenceFiles in HDFS creates records with the image identifier as the key, and the descriptor as the value.

Building the Tree of Representatives. We developed a Java implementation of the creation of the index tree containing cluster representatives. This tree of representatives is built outside Hadoop and serialized to a file subsequently used for clustering the data collection.

Creating the Index: Clustering. The clustering process assigning points to cluster representatives uses the index tree to efficiently discover the cluster each point belongs to. The Hadoop application consists of a map function that loads the tree of representatives and then reads the block of data it has to process. Each point from this block traverses the index tree until its closest cluster representative is known. Then the map task emits a *(cluster-id, point)* record and loops. The reduce function simply outputs in SequenceFiles the records received from the mappers. It is key to realize that the index tree is loaded by each map task at startup time. The tree will thus be loaded as many times as there are map tasks needed to complete an entire Hadoop job execution.

Searching Batches of Queries. A batch contains a very large number of query descriptors. Before being used to search the index, the query descriptors are reordered according to the identifier of the cluster each query points falls into. To do this, each query descriptor traverses the tree of representatives until it hits the bottom level, at which point the cluster identifier is known. We build a lookup table to keep track of these identifiers for every query descriptor. This table is created outside Hadoop and sent to every map task when a batch search is fired.

When spawned, mappers start by loading the lookup table. A mapper then receives its block of data. It then finds in its block the records having any of the cluster identifiers existing in the lookup table. Only those records are subsequently used for distance calculation. It is possible that not all records of a block are used because (i) it is unlikely all query points of one batch will fall into distinct clusters, (ii) there are typically much more clusters than query points in a batch, (iii) a block typically contains several clusters. Mappers emit k-nn results.

V.3.6. Index Creation

We have logged the performance results for running the index creation process on Hadoop, while increasing the data set from 10M to 100M images. We report here the results when indexing 20M and 100M images. Other results are in publications.

Exp. #1: Indexing 20M Images, 7.8B Descs., 1.0TB. This experiment indexes 20% of the full set, that is 1TB of data, about 20 million images and 7.8 billion descriptors. We used a total of 108 nodes from the three clusters. In this configuration, 3 nodes are dedicated to managing the system, leaving 105 tasktrackers nodes. The system is set to using, per machine, at most 8 slots for mapping and 2 for reducing.

With 108 nodes, it takes approximately 95 minutes to complete the indexing. Two comments are in order. First, the tree of representatives used to guide and do the assignment of points is quite large. It uses about 1.5M representatives. Not only this occupies a lot of RAM (461MB), but it also takes some time for each mapper to load from disks that tree into memory and to create the data structure for subsequent assignments. It also means quite a lot of distance calculations are needed to assign a descriptor as there are many representatives eventually guiding to a large number of clusters. Note also that the work achieved here is in part impacted by the uneven distribution of the representatives in the tree, from one level to the other. Therefore, a large fraction of the data traverses rather dense branches of the tree of representatives, requiring to do more distance calculations to find the closest representative guiding to the next lower level.

Second, the Hadoop architectural design decisions collide with the specific characteristics of our application: (i) the total number of map tasks to run is determined from the data collection size divided by the size of a block (512MB here, resulting in running about 2050 map tasks); (ii) the total number of map tasks is totally independent from the number of nodes used to run the entire job; (iii) a new map task is spawned every time a new block of data is to process; (iv) at spawning time, a map task has to load whatever auxiliary information it needs to correctly process the data in its block (in our case, the tree of representatives, 461MB to load every 512MB of data to index!).

Spawning a mapper thus includes a fixed overhead for reading the tree of representatives. This tree is loaded again and again, even by mappers running on the same physical node. Hadoop uses a distributed cache system, trying to cache locally to each machine such files. Nevertheless, loading again and again in memory that tree is time consuming.

Exp. #2: Indexing 100M Images, 30.2B Descs., 4TB. The second experiment indexes the full dataset using 108 nodes. With this configuration, the tree of representative is large as it uses more than 6 million data points to accommodate with the 30.2 billion descriptors to cluster. The tree occupies roughly 1.8GB in RAM.[3] This forced us to reduce the number of map tasks per machine to 4 only as otherwise not enough RAM was available for each mapper.

With this setting, it took about 10 hours to cluster the entire data set. A careful analysis of the logs shows that 99% of the reduce tasks where completed after 520 minutes, and the remaining 1% reduce tasks completed after 80 additional minutes. The reason behind this behavior is in part the uneven distribution of points to clusters.

But there is another explanation to this response time. Finely analyzing the data collection, we discovered that it contains hundred thousands of *identical* distracting images that turn out to come from a small set of explicit web sites having different URLs redirecting to a unique point. This is unfortunate, but it is a good example of what happens in the real world when

[3]Note 1.8GB of auxiliary info have to be loaded every 512MB of data to cluster, and this for 8000 Map tasks!

Parameter	Default value	Tuned value
#Map slots/tasktracker	2	8
#Reduce slots/tasktracker	2	8
Input data replication	3	3
Output data replication	3	1
Chunk size	64 MB	512 MB
JVM reuse	off	on
Map output compression	off	on
Reduce parallel copies	5	50
Sort factor	10	100
Sort buffer	100 MB	200 MB
Datanode max. receivers	256	4096
Namenode handlers	10	40

Table V.7.: Hadoop configuration tuning

indexing images. It would have been possible to filter these images but this would have required a specific ad hoc process. The direct impact of so much duplicates is that there is a small set of clusters into which the descriptors of these images accumulate, creating very large, unbreakable clusters, and writing them to disks takes a lot of time.

Exp. #3: Optimizing Hadoop. A careful analysis of the logs when indexing 4TB of data shows that several optimizations at the level of Hadoop's settings are possible. First, indexing 4TB means that about 16TB of data are shuffled around: the 4TB raw data exists in three copies for availability and performance, and the final indexed data occupies 4TB, not replicated, though. As that much data travels between nodes, it is worth considering compression to reduce any network bottleneck as well as reducing the load on disks. Since all the data that gets in also gets out, it is key to have large buffers as well as many parallel connections in order not to block data producers that run at a different pace than data consumers.

Table V.7 lists the various values of the optimized parameters for configuring Hadoop. Compression, for example, reduced the volume of data by 30%. Re-running the experiments above with 100+ machines and the 20M as well as the 100M images collections shows significant performance improvements. Indexing 20M drops from 95mn to 75mn, and indexing 100M images goes from 10h to 8h27mn.

Exp. #4: Adapting to Heterogeneous Hardware. It is never expected that all the machines in a grid are the same. Heterogeneity is common, and machines typically differ in their numbers of CPU, cores, their amount of RAM, disk space and possibly network capacities. This is clearly the case for the Grid'5000 testbed we are using, as this can be observed by reading the very different specifications reported in the Table V.5.

Hadoop does not take heterogeneity into account. It allows configuring users to define only one set up that must match with the specifications of the least equipped machine. Consequently, the framework is not using some resources on more powerful nodes, overall wasting resources. This is typically what we face when using Grid'5000. We can not naturally ask Hadoop to run more Map tasks on Cl_3 that has machines with 24 cores than it does on Cl_1 or Cl_2 that have only machines with 8 cores. Machines from Cl_3 are used only at one third of their capacities,

simply because Hadoop is unable to capture this hardware diversity. Note that RAM related settings are also concerned.

In order to fully exploit all the available power of our three clusters, we tricked Hadoop by imposing a two-step configuration. We first configure and deploy Hadoop using a global least common denominator setting. We then create cluster specific config files and we force tasktrackers to reboot and use these files.

We then ran some experiments to measure the improvements. One experiment reported here uses the default settings assigning 8 map slots and 2 reduce slots per node for all three clusters. Another experiment applies these settings to Cl_1 and Cl_2 only, and sets 20 map slots and 4 reduce slots for the machines in Cl_3 after reboot. When indexing 1TB of data, then the default settings are such that Hadoop terminates after 95 minutes, while the heterogeneous settings make Hadoop terminate indexing after 65 minutes, 30 minutes faster.

V.3.7. Batch Searching

This section reports the performance results obtained when searching the full data collection with batches of query images. The images in the batch are the 3,055 variants from the Copydays evaluation set—this query set is the most challenging. The results are expressed both in terms of response time and search quality. Response time wise, we record the time it takes to complete the query batch using the 100+ nodes. Quality wise, we search for the 20 nearest neighbors of each query point computed from the query images. There are just below 1M query points in the batch. Each nearest neighbor votes for the image from the indexed collection it belongs to, and the votes are aggregated to eventually return the identifiers of the most similar images. We have a rather strict success criterion for searching: the search is said to succeed if and only if the original image identified from its query quasi copy has rank 1; the search is said to fail otherwise. The percentages given when discussing quality thus correspond to counting the number of times original images are ranked first.

The lookup table built from the descriptors in a batch (see Section V.3.5) is stored as an HDFS file read by all search mappers when they are spawned. It takes about 3 minutes to build this lookup table on a single core outside Hadoop. The lookup table file is replicated three times to reduce contention when mappers access it. 100+ machines are used here.

Exp. #5: Searching, Time and Quality, 100M Images, Default Settings. In this first experiment, the block size for the indexed data is 128MB, with hence 33,483 mappers to run. The configuration of Hadoop is not optimized—we therefore have a baseline for comparing the performance. Note that the quality performance are independent of such settings, only time is impacted.

Searching the entire batch took 1,623 sec. on average, or just over 27 minutes. This gives an average processing time per image of under 530ms. Figure V.4 shows the quality results of the search. This figure plots for every family of variants the percentage of original images found at rank 1. It also plot the average percentage across all variants at the far right end of the figure. Note for comparison we also measured the quality when indexing 20% of the data collection. From the figure, it is clear that DeCP returns high quality results, except for some severely attacked images such as when 80% of the image is cropped and then it is rescaled to its original size, or when strong manual variants are applied. Note that we count as search failures the cases when no descriptor can be computed on the query images (this happens for 6 variants). It is interesting to observe search quality does not significantly degrades when the size of the

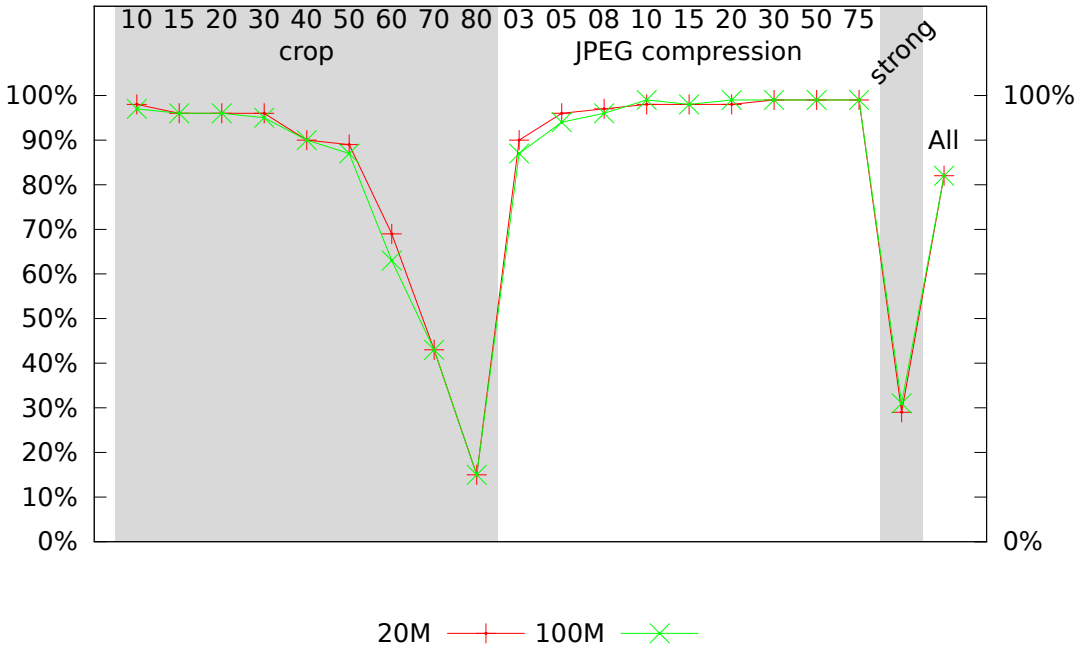


Figure V.4.: Search quality, Copydays evaluation set

distracting dataset increases. Overall, 82.68% of Copydays variants are found when drowning them in 20M images, and we find 82.16% of them when drown in 100M images.

We have also ran quality experiments using the 49k query set. We find 91.65% of the variants at rank one. This is an excellent score.

Exp. #6: Searching, Time, Varying Batch Size, Tuned Settings. In this experiment, we have been checking the time it takes to search the complete Copydays batch (approx. 3,000 images) as well as another much larger batch containing 25,000 images. We used here 100+ machines configured using the tuned parameters presented above. Here, it takes about 382 seconds to complete the Copydays batch, this is 127 ms per image on average. The same experiment, with the untuned Hadoop takes a little more than 500s. It takes about 755 seconds to run the 25,000 image batch, this is 30 ms on average per image. We certainly observe here a similar phenomena as the one highlighted above in Section V.2.6 (multicore batch searching). When a batch is small, then the response time is limited by the time it takes to read the entire indexed collection. Then, as the batch increases in size, there is more and more overlap between the I/O requests and the CPU computing distances, overall improving throughput and reducing the time for processing one image. Eventually, for extremely large batches, the cost will be dominated by the CPU and the time for processing each image will stabilize, and possibly increase if thrashing. We have not reached this point yet but conjecture it will happen.

The current implementation of the batch search runs into RAM problems because we load the entire lookup table in memory. For extremely large batches, this table gets big and forces us to reduce the number of cores we use per machine in order to give each core more main memory. The current implementation does not cope with batches as large as the ones we used when we discussed the multicore batch search, Section V.2.6.

Task	Duration	Approx. %age
Environment deployment	10 min	3%
Hadoop deployment	5 min	1.5%
Data transfer to HDFS	90 min	28%
Index creation	170 min	53%
HDFS optimal chunk placement	30 min	9%
Lookup table creation	3 min	1%
Searching	5 min	1.5%
Retrieving search results	5 min	1.5%

Table V.8.: Time measurements for workflow steps

V.3.8. Observations

The work we have been doing for distributing DeCP is still in its early stages. We have many other parameters to check, and quite a lot of understanding to gain from analyzing experimental results. There are few lessons we can however already draw from our work.

Deployment Overhead. The deployment overhead for each experiment is substantial when running at large scales. Since the grid is a shared tool, it operates using reservations that allow users to employ resources for a certain time slot. After the reservation expires, all deployment data and setup are deleted. Consequently, creating the experimental environment, setting up the Hadoop cluster and making the data available in HDFS have to be repeated for every experiment.

A typical experiment involving indexing and searching 1 TB of data on 50 Grid'5000 nodes requires a time frame of 5-6 hours. To better analyze this time-consuming process, we divide it into several steps and we provide the amount of time allocated to each step, see Table V.8.

The first step of the workflow accounts for deploying and configuring the execution environment: creating an isolated environment on Grid'5000, starting Hadoop processes, launching monitoring tools to collect information about the platform usage and job statistical data. As Table V.8 shows, a substantial amount of time is spent on copying the data from local storage to HDFS.

Node Failures. Node failures represent the daily norm in grid environments. Even though Hadoop is designed to cope with failures in a transparent manner, machine deaths can severely impact the whole deployment. On Grid'5000, we experienced from one to five node failures during a 60 hours run, some failures requiring a complete re-deployment to exclude the failed nodes. The worst failure outcome is losing the data on the machine. To avoid this, we used a replication data factor of 2 or 3. However, this is not always possible for very large datasets; factors such as storage, replication time, add substantial costs.

Independence of Map Tasks. Hadoop's map tasks are completely independent and each require to load the tree of representatives. When this auxiliary information is large, then each map task consumes a significant portion of the RAM available on a node. This, in turn, means that map tasks are unable to run inside every available core in a node, because there is not enough RAM. It is unfortunate to waste some of the processing power, leaving cores idle because there is no way to share data, even *read-only data* (as is the tree of representatives) between map tasks running on the same node. This observation suggests for application programmers to implement multi-threaded map tasks. This is more complicated to program but it is one option for using

all the processing power of nodes while circumventing Hadoop's inflexible architecture. With multi-threaded map tasks, a single task would load the auxiliary data only once and then would process its block of data faster thanks to its multiple threads running on multiple cores. In the case of the experiment with the full 100M images data set, one single map task could then use up to 6 threads processing data in parallel on Cl_3, overall keeping the 24 cores constantly busy, instead of using only 4 cores now.

Large Auxiliary Monolithic Data. Performance can possibly be hurt when two conditions are met: (i) the data collection occupies *many* blocks, hence many map tasks have to be run, and (ii) each map task need to load at once *a lot of auxiliary information* at startup time. This is exactly what happens at index creation time. Each mapper has to load the entire tree of representative in its memory. It is monolithic data, it can not be split in parts such that each part is loaded independently. Mappers need the whole data structure. This causes some severe overhead, especially when blocks of raw data to process are small compared to that auxiliary data. In some of the settings presented above, the tree of representatives could occupy close to 2GB, and loading this tree every time a mapper processes a 128M block is awkward.

It is key to reduce as much as possible the overhead payed by each map task at spawning time. One possible option is to increase the size of the blocks of data to a value that is significantly larger than the ones recommended by Hadoop that are typically 64MB or 128MB. Setting this to 512MB or few GB in turn reduces the number of map tasks to spawn and thus reduces in proportion the time wasted when each map task starts. Note, however, that big data blocks may cause some nodes to run out of disk space as the temp area buffering the data produced by mappers and consumed by reducers fills up faster when blocks are big.

Large Auxiliary Splittable Data. In some cases, the auxiliary data to load at Map time is not monolithic and can be split into smaller parts that are loadable separately. This is for example the case for batches of queries. So far, the implementation described above simply loads the entire query batch and then does the search. When the batch is large, then a lot of data has to be loaded again and again, and indeed only a small portion of it is relevant to the cluster scrutinized by each mapper. As it was highlighted above, large batches might force to reduce the number of mappers per node, hence reducing performance. Hadoop proposes a mechanism called *MapFiles*. When using MapFiles, Hadoop hashes an entire file according to the values of the key field from each record, and group colliding keys into the same bucket. At run time, a Map task loads the index of this hash table and then subsequently solely fetches the bucket(s) that it needs for processing. This saves a lot of memory since only the necessary part(s) are consuming memory. Investigations demonstrating the usefulness of this mechanism are on the way.

Replication Helps Performance. Most of the mappers read the data locally, only about 1% of remote reads were observed. This is again the case for setting the replication factor to a value above 1. We have not seen any major performance improvement for any value > 2. In contrast, maintaining a single copy of the indexed data causes roughly 8 to 10% remote reads, hurting performance. Rack awareness and replication are good for performance, and not only for coping with failures. Note that we experienced several nodes failures and happily observed that Hadoop re-ran tasks, eventually completing the runs.

V.3.9. Discussion

Map-Reduce, and its open source implementation, Hadoop, are very helpful for running computations on very large datasets. Failures, scheduling and other complicated issues are transparently handled by the framework. Most applications that successfully run with Map-Reduce and Hadoop are however built around an *acyclic* data flow model. This model is not suitable for some other applications that for example reuse a working data set across multiple parallel/ distributed operations. The popular k-means, almost all clustering algorithms, including outlier detection, and more generally a quite large body of machine learning techniques are iterative by nature. Implementing them on top of Map-Reduce and Hadoop turns to be extremely inefficient. Interactive applications are also quite hard to achieve with this framework. The startup time is large, and batch processing is the de facto paradigm. While this favors throughput, some applications need short response time. With Map-Reduce and Hadoop, no exploration, no navigation inside a dataset is possible.

Basically, Map-Reduce and Hadoop do not cache anything between runs. To use the data produced by a previous run inside a subsequent run, data must be pushed to an external storage system. This incurs a substantial overhead.

Mahout [Owen et al., 2011] is one popular machine learning library that uses Hadoop as its execution engine. Iterative tasks as k-means, in Mahout, require to use an external driver program that submits multiple jobs to Hadoop. Each job fully starts an entire Hadoop execution engine, where mappers and reducers are spawned and do the work. The overhead for starting up the entire Hadoop environment is therefore payed multiple times. The driver launches jobs producing data on disks, read again at the client side to check for convergence, and possibly initiate another round on execution if needed. This driver program executes logically and sometimes physically outside Hadoop, and consequently does not benefit from transparent fault tolerance handling. There has been some proposal to fix some of these problems with for example Twister and Pregel. Twister [Ekanayake et al., 2010] and Haloop [Bu et al., 2010] are iterative Map-Reduce runtimes. They make possible some form of data sharing between runs. Twister, however, has no form of fault tolerance; Haloop can not cope with interactive tasks.

Spark [Zaharia et al., 2010] is a new framework recently proposed and which addresses both interactive and iterative computations on clusters of computers. Spark retains the scalability and fault tolerance of Map-Reduce. Data in Spark is partitioned across machines, and lost data can be rebuilt through the notion of lineage. Spark might be worth trying as it may help determining a better visual dictionary for eCP: few rounds of k-means are likely to greatly improve the quality of the index tree, which, in turn, will improve the quality of retrievals. Spark might also help to address response time issues while DeCP has been targeting batch processing to match Map-Reduce constraints. Having a system that can optimize for response time and throughput might be possible with Spark.

V.4. Concluding Chapter V

By construction, the Hadoop framework stores data on disks, resists failures because data is replicated, it copes with scale, obviously. It somehow fits with some elements of the database perspective we sketched in Section I.4. Yet, it is not entirely satisfactory. We highlighted in the discussion preceding this conclusion some of the flaws of Hadoop. More generally, working at large scale with a CBIR is already quite complicated, and the extra work for setting a framework such as Hadoop is a real pain.

This is particularly true since we are still confined inside a research lab type-of working mode. For good scientific reasons, we decide that is it relevant to fire a particular experiment with these specific settings to try to understand this or that. Then we do it, collect the result, halt the system until next time, where we have to redo again the entire process. We are not yet where an constantly distributed running system is there, ready to process queries and/or batches. Therefore, it is hard to meet some of the objectives we defined such as handling dynamic updates, recovering from media failures, ... Even though Hadoop claims it copes with failures, injecting faults and studying the impact of the behavior of a CBIR system would be interesting.

This chapter is mostly the result of the work Gylfi Þór Guðmundsson [Guðmundsson, 2013] did during his Ph.D., with help from Diana Moise and Denis Shestakov. While that work is focused on grid of computers, Gylfi also questioned the use of peer-to-peer systems as a possible alternative for designing a distributed CBIR system working at very large scale.

Aside this alternative, having worked on large scale content-based image retrieval for a decade allows us to draw some perspective to this work. These perspectives are discussed in the next chapter.

Chapter VI

Perspectives

Contents

So far, we have been discussing techniques for building efficient content-based image retrieval systems managing extremely large databases of photos. It is now time to turn to discussing perspectives to our work. The perspectives presented in this chapter follow research paths that are not all fully aligned with what has been discussed in the previous chapters; the perspectives are not a direct continuation of our previous work. This chapter starts with presenting a new mechanism aimed at improving the quality of similarity search results. Then, we describe a data model facilitating organizing and browsing multimedia collections. We finish by considering the security and the privacy of multimedia contents. We briefly overview these perspectives in this introduction before discussing each of them in greater details.

The first perspective we discuss here is about finding means to improve the quality of the results returned by a CBIR system. At scale, so far, the only way to efficiently identify similar elements is to run approximate searches. This has been a leitmotiv in this manuscript. Approximate searches obviously impact quality. Geometry in images is often taken into account to improve the quality of the final result since it allows to remove some false positives. But such traditional means have been extensively covered in the literature. Alternatively, it is however

possible to address quality issues by revisiting the more fundamental k-nn search process that determines the neighborhoods of query points. Indeed, quality is directly related to neighborhoods. A noisy neighborhood will in turn cause quality to drop, and quality improving techniques such as geometry can only provide little help here since they start from poorly relevant information. Instead, the quality of the results can be improved by properly accounting for the inherent asymmetry of k-nearest neighborhoods. Compensating that asymmetry and considering reciprocal nearest neighbors and/or shared nearest neighbors improves result quality. We have started investigating this perspective, which is presented in much more details in Section VI.1.

The second perspective that we describe in this chapter is about organizing and browsing multimedia collections. Much of the research in multimedia is trying to extract information from documents, and that information is then indexed, structured, classified, etc., before being enrolled in a search engine. In contrast, linking multimedia information is an emerging topic, and there is an increasing number of publications describing techniques to thread news reports, to extend the concept of hyperlinks to multimedia data, etc. Overall, what seems to emerge is that research in multimedia becomes now not only concerned with items (images, faces, objects, ...) but also with the relationships between items. So far, no clean model has been proposed to appropriately represent multimedia data as well as the relationships between data items. It turns out that the database community proposed a formal data model to facilitate making sense of vast amounts of traditional data as well as to nicely grasp the relationships that exist between data items. This model is called the multi-dimensional analysis model, or MDA, and it has been instrumental for running on-line analytical processes. This second perspective is about understanding what becomes possible when applying the concepts of the MDA model to data extracted from multimedia contents. It is presented in Section VI.2.

The third perspective presented in this manuscript is related to the *securvacy* of multimedia contents, a term referring to both the security and the privacy dimensions of CBIR systems. Such systems are now used as a filtering tools, for example to fight against the piracy of multimedia contents. Many publications in the last few years have proposed very robust schemes where pirated contents are detected despite severe modifications. None of these systems have addressed the piracy problem from a security perspective, and very little work addresses privacy issues at scale. Section VI.3 is a long discussion about the many security and privacy problems that CBIR systems have to address.

The three above perspectives have already received some substantial attention from us. Some of the issues described in the subsequent sections have already materialized into publications; all issues are currently under active study, and some publications are pending. Finally, there are few other perspectives that are still in their infancy and for which we only provide little details. These perspectives are grouped together in a section that is concluding this chapter.

To facilitate reading this chapter, a page break is inserted between each perspective.

VI.1. Improved Quality with Shared and Reciprocal Nearest Neighbors

This section is discussing the use of shared and reciprocal nearest neighbors to improve the quality of similarity search results. We first introduce the problem and motivate the need for using neighborhood information. We then describe some early research results as well as some initial proof-of-concept experiments before presenting several research directions for this work.

This first perspective comes from the work Agni Delvinioti did during her internship. This work materialized into a publication, see [Delvinioti et al., 2014]. Please note also that much credit for this first perspective is to be given to Michael E. Houle and Hervé Jégou, in addition to Agni.

VI.1.1. Introduction

At scale, approximate searches rule, but tend to lower quality. In fact, quality reduction is caused by at least the conjunction of two design choices. The first choice concerns the use of quantization, the de facto technique used for indexing, and this is a lossy process. The second choice concerns the decision to analyze a small number of cells to further save costs. This, in turn, might prevent the searching procedure to identify the correct neighbors since some points are not scrutinized.

In the case of images, various techniques have been designed to enhance the quality of the result returned to the user. Many of them adhere to the following simple principle: a first series of candidate images is determined using a *primary* similarity criterion, typically a k-nn search computing Euclidean distances or checking the cosine similarity of points. These images are temporarily kept in a shortlist. Then, the images in that shortlist are re-ranked according to a *secondary* similarity measure before being returned to the end user.

A very successful re-ranking scheme relies on comparing the geometrical consistency between the images in the short list and the query image. The assumption is that highly geometrically consistent shortlisted images are likely to be more similar to the query image than are shortlisted images having their geometry poorly consistent with the query.

The consistency of the geometry between a pair of images is typically determined by first estimating the parameters of a mathematical model transforming one image into the other, and then determining how many points in the images fit the model. The more points, the more geometrically consistent are the images. The seminal approach materializing this idea is the RANSAC algorithm by Fischler and Bolles [Fischler and Bolles, 1981]. Its cost is high, however, since it requires heavy computations, especially when the model has for example 6 degrees of freedom to cope with affine transforms. Using a Hough-Transform [Szeliski, 2010] before running RANSAC diminishes the cost, that however remains high. Other lighter approaches have been proposed where geometry is used in a weaker manner, trading off the consistency checking against dramatic costs reductions, see for example [Jégou et al., 2008a].

Note that other approaches have also been defined to improve the quality of the results, such as better defining the (larger) visual vocabulary [Nistér and Stewénius, 2006], by altering the distance measure itself [Jégou et al., 2007], or by aggregating the local features in clever ways [Perronnin et al., 2010, Jégou et al., 2010a].

Overall, all these approaches do improve the quality of the results, yet, there is still room for improvement.

Recently, however, another series of contributions proposes to enhance the quality of the result by relying on the use of shared and reciprocal nearest neighbors. This section discusses some initial works we did in this direction. More details, deeper explanations can be found in [Delvinioti et al., 2014].

VI.1.2. Motivation

One significant impediment to neighborhood-based similarity search is the asymmetry of the k-nn criterion used to identify similar items. If x is the among the k nearest neighbors of y, it is not necessarily the case that y be among the k nearest neighbors of x. In other words, reciprocity is not guaranteed. When this happens, although x and y are judged to be similar according to the measure used to establish the neighbor rankings with respect to y, from the perspective of x, many other objects have been judged as being of greater relevance. Thus, a lack of reciprocity in k-nn relationships can be taken as an indication that query result sets are likely to contain many noisy data points of low relevance to the query. The presence of many such false positives may drastically lower the quality of the overall query result.

VI.1.3. Related Work

In an attempt to improve the quality of relevance rankings of query results, several approaches augment k-nn ranking criteria with information derived from the neighborhoods of multiple points. For example, Qin et al. in [Qin et al., 2011] re-rank elements according to the notion of k-reciprocal nearest neighbors, based on an initial ranking in terms of the cosine similarity measure. They basically apply different distance measures to different parts of the shortlist. Given a query image q, they separate the database into two disjoint sets, the *close-set* containing images highly related to q, and the *far-set* comprising the remainder of the database. The *close-set* is used to re-rank images from the *far-set* according to the degree of connectivity of *far-set* images to *close-set* ones.

While the method is able to take advantage of the highly related neighbors within the *close-set*, when ranking elements of the *far-set*, the use of arbitrary threshold values on the (non-reciprocal) primary cosine similarity ranking leads to unstable performance behavior as the neighborhood size k increases. Furthermore, the construction process for the *close-set* is quite costly. Overall, their set partition strategy for exploiting structural information over neighborhoods turns out to be insufficient for practical applications on image data.

In another context, Houle et al. in [Houle et al., 2010] propose the use of shared nearest neighbor information for re-ranking. Experimental evidence indicates that shared-neighbor measures are more stable and robust than traditional noisy approaches, especially in high-dimensional spaces. Here, the similarity value of an object pair is a function of the number of data objects in the common intersection of fixed-sized neighborhoods, determined by a conventional (primary) similarity measure. The primary similarity measure can be any function (L_2, cosine) ranking the data objects relatively to the query.

The *relevant-set correlation* clustering model [Houle, 2008] adopts such a shared-neighbor scheme to account for well-associated items in the grouping procedure. [Hamzaoui et al., 2014] also builds on shared-neighbors. They designed a bipartite shared-neighbor clustering algorithm for suggesting additional object-based visual queries suitable for relevance feedback search.

All three of the aforementioned approaches rely heavily on some notion of shared neighborhood. Each attempts to compensate for the difficulties inherent in high-dimensional search, by extracting adjacency and structural information among neighborhoods within the image space,

or among object seeds within the object space. The three methods all can be regarded as attempting to "denoise" the primary traditional similarity measure, in a second processing phase, in order to provide more reliable matches.

VI.1.4. Initial Steps

We made initial steps in this direction, trying to explore what are the improvements in the quality of the results when using reciprocal nearest neighbors and/or when relying on shared nearest neighbors. Our two initial steps aim at compensating for the fundamental asymmetry of the k-nn measure. The first step builds on the definition of three robust and stable extended similarity measures for comparing the neighborhoods of the candidates in the shortlist, before re-ranking them. The second step builds on the definition of a maximum reciprocal rank criterion to construct a better shortlist containing more highly relevant images. When used either in isolation or in a combined manner, these two steps significantly improve the accuracy of image search engines when evaluated against traditional benchmarks.

We briefly present these steps below.

Three Shared Nearest Neighbors Metrics

In traditional systems, for a query image q, a shortlist $\mathcal{N}_k(q)$ of results is produced from a k-nn candidate set, where the membership and order is determined according to a similarity measure defined in advance. It is possible to re-rank the images in the shortlist by considering the number of similar images that are shared by the members of the shortlist. This re-ranking strategy borrows from the notion of shared nearest neighbors studied by Houle et al. [Houle et al., 2010]. Once a shortlist $\mathcal{N}_k(q)$ has been determined, it is then possible to parse it to determine the relationship between the neighborhoods of any two of its elements. Given two images x and y in the shortlist of q, their shared neighbor set is defined as the number of images in the common intersection of their k-nn sets. The shared neighbor set is more formally defined as

$$SNN_k(x, y) = \mathcal{N}_k(x) \cap \mathcal{N}_k(y);$$

Information concerning the pairwise relationships among all images in the shortlist can in principle be incorporated into a similarity measure for the purpose of re-ranking that shortlist. Two images in the shortlist that share many database images are likely to be more similar than two other shortlist images sharing few relevant images. Comparing the neighborhoods of the images in the shortlist can therefore serve for the comparison of the images themselves.

Base Metrics. We have been playing with three measures for computing the similarity between two neighborhood sets:

- *Jaccard:* A traditional measure of the similarity between sets is the Jaccard coefficient:

$$j_k(x, y) = \frac{|SNN_k(x, y)|}{|\mathcal{N}_k(x) \cup \mathcal{N}_k(y)|}$$

- *Set Correlation:* Another possible measure is the set correlation measure defined by Houle [Houle, 2008]. For x and y in database $\mathcal{D}$, this measure is:

$$sc_k(x, y) = \frac{|\mathcal{D}|}{|\mathcal{D}| - k} \left(\frac{|SNN_k(x, y)|}{k} - \frac{k}{|\mathcal{D}|} \right)$$

- *Sigmoid:* The Jaccard and Set Correlation measures both fail to differentiate between the case where strong (original) similarity scores are observed when k is small, from the case where weaker similarities are observed when k is large. For the comparison of neighborhood sets, a sigmoid function can be used to differentiate strong from weak similarities while mitigating the influence of (large) k. The function we define is, the slope of the curve being $a = 1$:

$$sgm_k(x, y) = \frac{1}{1 + exp(-a * (\frac{|SNN_k(x,y)|}{k} - b))},$$

Extended Metrics. The Jaccard and the Set Correlation measures are sensitive to the membership and sizes (k) of the neighborhoods, but not to their order; as such, they are oblivious to the rank at which neighborhoods begin to greatly diverge. To take this into account, we extend the above measures. Each extension can be viewed as voting processes retaining reliable high-quality votes that are likely near the top of the shortlist (with initial ranks closer to 1), as well as accounting for divergence in neighborhoods based at items at the bottom of the list (with initial ranks closer to k). The extended measures are:

- *Extended Jaccard:*

$$\overline{j}_k(x, y) = \sum_{k'=1}^{k} j_{k'}(x, y) \frac{1}{\sum_{l=1}^{k'} \delta_l(x, y)},$$

with[1]

$$\delta_l(x, y) = \begin{cases} 1, & \text{if } |SNN_l(x, y)| > 0 \\ 0, & \text{otherwise} \end{cases}$$

- *Extended Set Correlation:*

$$\overline{sc}_k(x, y) = \sum_{k'=1}^{k} \frac{sc_{k'}(x, y)}{k'}$$

- *Extended Sigmoid:*

$$\overline{sgm}_k(x, y) = \sum_{k'=1}^{k} \frac{sgm_{k'}(x, y)}{k'}$$

Maximum Reciprocal Rank

In [Qin et al., 2011], Qin et al. use a reciprocal k-nn criterion for their *close-set* and then re-rank images. In contrast to their approach, which can be rather costly and unstable, we propose here a new re-ranking criterion based on reciprocity of k-nn set membership.

We first define $\text{rank}_x(y)$ as the rank of the image y when the database is queried with x, according to some underlying primary similarity measure (possibly but not necessarily the cosine similarity). Conversely, $\text{rank}_y(x)$ is the rank of x in the query result determined from querying with y. From the perspective of x, $\text{rank}_x(y)$ will be referred to as the *forward rank* of y, while $\text{rank}_y(x)$ is termed the *backward rank* of y. We then define the following reciprocity-based symmetric dissimilarity measure:

$$r(x, y) = \max_{x,y \in \mathcal{D}} (\text{rank}_x(y), \text{rank}_y(x)).$$

[1] Care is taken to avoid divisions by zero.

Based on $r(.,.)$, we define $\mathcal{R}_k(x)$, the k-Maximum Reciprocal Rank Set of an image $x \in \mathcal{D}$, to be the k items of $\mathcal{D}$ achieving the smallest maximum reciprocal rank values in conjunction with x:

$$\mathcal{R}_k(x) = k\text{-arg}\min_{y \in \mathcal{D}} r(x, y).$$

$\mathcal{R}_k(x)$ identifies images in the vicinity of x having a high degree of mutual relevance: these images are not only reciprocal neighbors of x, but the extent of reciprocity is strictly bounded by the value of k. This reciprocity-based neighborhood is a much stronger indication of mutual similarity than it can be determined by an asymmetric k-nn primary similarity measure.

For these reasons, we propose the use of the re-ranked set $\mathcal{R}_k(q)$ as a substitute for the original neighborhood set $\mathcal{N}_k(q)$ when determining the contents of the shortlist corresponding to the query q.

VI.1.5. Initial Performance Measurements

We have used three popular datasets to evaluate the impact of our schemes on the quality of results. These datasets are (i) the Holidays image set ([Jégou et al., 2008a]) indexed using a 200k visual vocabulary , (ii) the Oxford5k image set ([Philbin et al., 2007]) assigned to 1M visual words and the Paris6k image set ([Philbin et al., 2008]) assigned to 500k visual words. The baseline for the evaluations is obtained by computing the mean average precision (mAP) on the results.

Three main lessons can be drawn from these experimental results:

1. *Shared Nearest Neighbor Information is Useful.* Our results show that taking into account the neighbors shared by the images in the shortlist dramatically improves quality. Spectacular improvements are with the Paris6k image set.

2. *Extended SNN Measures Work Best.* Integrating the values of measures over a range of neighborhood sizes can boost performance of re-ranking methods. Another effect of this extension of re-ranking measures is that the performance is more robust to increases in the value of k.

3. *Reciprocity Helps.* Building the shortlist using the Maximum Reciprocal Rank clearly improves over the use of the original k-nn sets.

VI.1.6. Research Directions

Several research directions emerge from this work.

Research Direction #1: Constructing a Nearest-Neighbor Graph

Our method requires the computation and storage of ranked neighbor lists. When the data collection is very large, then this step is really problematic. So far, there is no good solution for constructing a k-nn graph at scale. The cost for brute-force constructing a k-nn graph is $O(n^2)$ where n is the size of the data collection. This is only practical for small datasets.

Recently, however, two rather efficient k-nn graph creation techniques have been published. Dong et al. in [Dong et al., 2011] propose an algorithm that is building an approximation of the true k-nn graph connecting high-dimensional data points. The algorithm starts by creating a very rough approximation of the k-nn graph as it randomly assigns k neighbors to each data

points. Then, it iteratively improves this approximation by exploring each point's neighbors' neighbors as currently defined. For one particular point, if one such (indirect) neighbor is closer than any of its current (direct) neighbor, then the list of k-nn associated to this point is updated. By repeating this strategy a certain number of times (6 times, which is not a lot in fact), the k-nn graph quickly converges and becomes a pretty good approximation of the true graph. Complexity is empirically observed to be around $O(n^{1.14})$.

Wang et al. in [Wang et al., 2012] have a very different approach. Their algorithm basically partitions the entire data collection into small groups, brute-force builds the k-nn graph within each group and then merges the many subgraphs. Please note that this last step is likely to be quite problematic as partitioning is complicated in high-dimensional spaces (that is, many edges are likely to exist between partitions).

Overall, these two aforementioned approaches have scalability limitations due to memory problems. Both approaches store the graph of k-nn as adjacency lists, and when the data collection is very large and/or when k is large enough, there is not enough RAM to accommodate the lists. In addition, Wang et al. use a global H-table to avoid computing several times the distance between the same pairs of data points, and this does not scale since the number of possible pairs grows rapidly.

So far, there is no satisfactory graph construction method that is efficient enough at very large scale. Inventing such methods is future work.

Research Direction #2: Storage Requirements for Large Nearest-Neighbor Graphs

Without loss of generality, representing very large graphs is still an open problem. In the case of Web graphs, Boldi and Vigna (see [Boldi and Vigna, 2004]) proposed a very compact representation of graph that exploits the inner redundancies of the Web in order to store a Web graph in memory in a limited space. Their scheme allows to access a compressed graph without actually decompressing it, using lazy techniques delaying the decompression until it is actually necessary. They have applied their technique to extremely large graphs comprising over than one billion node and six billion arcs.

In contrast to Boldi and Vigna who are focusing on Web graphs, Neo4j is an open-source general graph database supported by Neo Technology (see [Robinson et al., 2013]). Neo4j claims to be a highly scalable, robust (fully ACID) native graph database.

It is of high interest to wonder how these two low-level foundations for playing with huge k-nn graphs would perform when applied to large image collections. Understanding the underlying model that Neo4j is using, as well as determining whether the inherent redundancy in graphs Boldi and Vigna need in order to efficiently compress their data structure exists in general k-nn graphs are part of future work.

VI.2. Organizing and Browsing a Multimedia Collection

Traditionally, research in multimedia has focused primarily on analyzing and understanding the contents of media documents, by defining clever ways to extract relevant information from the multimedia files, thereby hoping to eventually bridge the semantic gap.

Today, data is everywhere. Collections of digital media are extremely large, containing billions of still images, hundreds of thousands of hours of videos. Beside media analysis, accessing such large volumes is complicated and storing and retrieving data to and from disks while guaranteeing performance requires some specific expertise. This has been covered in this manuscript. But there are others problems that are currently addressed by the multimedia community.

One of them is the browsing of multimedia collections, which is discussed in this section. This perspective owes much to the bilateral long lasting cooperation with Björn Þór Jónsson as well as to other people who have contributed to this project: Giacomo Cesca, Áslaug Eríksdóttir, Kári Harðarson, Gísli Kristján Ólafsson, Mirco Pazzaglia, Kristján Rúnarsson, Hlynur Sigurþórsson, Grímur Tómasson and Ólafur Waage.

VI.2.1. Current Problem: Multimedia Data is not Information

We have observed that much of the research in multimedia is trying to *link* the information automatically extracted from the contents to create a meaningful user-experience. Threading the contents of the news reports to follow the various developments of a particular news across time; structuring TV broadcasts in shows, commercials, films, and automatically enriching that contents with hyperlinks; automatically classifying pictures according to a family tree; these are all typical examples of problems where extracting contents is as important as associating pieces of contents that have some relationships.

Linking information automatically extracted from multimedia data is a new trend in the community. That work was not possible few years back because the research in multimedia was dedicated to extracting and then indexing contents, this being required before being capable of linking related data elements. Now, we have rather mature solutions to extract various information from images, from videos, from audio tracks, etc. It is largely admitted that using all that information, from multiple modalities, as well as weaving links between information pieces is helping getting semantics from contents. Linking contents is now attracting researchers.

It turns out that the multimedia community is pursuing that goal of linking contents in a very ad-hoc manner, each paper proposing its specific data model, its own ways to link data items. Overall, the trend we observe suggests that state-of-the-art approaches are quickly overwhelmed with multimedia data. We believe that multimedia is lacking a powerful and flexible data model where multimedia data (ranging from entire documents to elements automatically extracted from the contents such as faces, scenes, objects, ...), as well as the relationships between data items can be appropriately represented.

VI.2.2. An Analogy: Business Intelligence

Methods to avoid being overwhelmed with data have long been studied in the database community. If we take a short historical perspective, then, traditionally, early relational database vendors provided users with tools to get back specific data via SQL-queries. Typically, users knew what data they were looking for and simply retrieved the actual values. For users wishing

to *understand* the data, however, by discovering trends and patterns, or via getting an overview of key figures, using the relational model was very painful; specifying SQL-queries was quite difficult, sometimes impossible, and the execution costs were enormous.

It therefore became crucial to invent an adequate framework to facilitate such on-line analytical processing (OLAP), enabling decision support and business intelligence applications. In order to help users *turn their data into information*, a formal data model was created, namely the multi-dimensional analysis (MDA) model.

The MDA model introduced two key concepts that revolutionized users' perception of data. These two concepts are *dimensions*, including hierarchies, used for specifying interesting sets of data, and *facts*, or numerical attributes, which are aggregated for an easy-to-understand view of the data of interest. The MDA model has proven to work extremely well in practice and has largely been accepted as the means for understanding huge (terabyte-sized) data sets [Connelly and McNeill, 1999].

The MDA model was instrumental in helping users make sense of vast amounts of numerical data. The model is implemented through *browsers* that help users discover their data collections without bothering them with physical details such as their access paths, ... These browsers are of great practical interest because they focus on the value of data items as well as on the relationships that exist between data items, abstracting physical considerations.

VI.2.3. Upcoming Work: Multi-Dimensional Media Browsing

The goal of this research is to understand what becomes possible when applying the concepts of the MDA model to data extracted from multimedia contents. While the MDA model is not immediately suited to model multimedia data, we believe that the similarity of the problems is significant enough to use the concepts of that model as the base of a powerful and flexible solution to multimedia data modeling.

Note that verifying the soundness of applying the MDA model to multimedia data in general is extremely difficult, as multimedia is very diverse, and core operations manipulating multimedia data (searching by similarity, classifying, quantizing, segmenting, running machine learning tasks, ...) have no counterpart in the OLAP world.

We have therefore decided to start exploring these issues from the restricted perspective of browsing personal photo collections. This restricted problem is much more simple, yet, it requires to define the foundations of a data model inspired by MDA and geared towards multimedia. Furthermore, a personal photo collection trivially contains many data items that it is very appropriate to link and organize in meaningful ways, such as people, places, events, dates, objects. It obviously includes a visualization aspect that is needed not only to browse pictures, but also to specify the relationships existing between data items and/or to correct the errors made by the automatic information extraction processes.

We therefore think bootstrapping our research from this perspective is key for rapidly making progress. We believe that we lose not so much generality through this restriction, but only gain by having a clearly focused application area. Once we will have gained a significant understanding of what it at stake when applying the MDA model to photo browsing, then we will try to go more general, first considering media browsing and then moving towards applying the MDA model to multimedia data in general.

VI.2.4. Initial Steps: Multi-Dimensional Photo Browsing

One of the motivations that gave birth to this work comes from the observation that the current set of photo browsers that are shipped with computers are very poorly helping users getting the most out of their personal photo collection [Harðarson and Jónsson, 2007]. Browsers are mostly geared towards helping users organize their images in hierarchies of folders, which essentially provide a GUI for the physical layout of files on disks. While this is both simple and familiar, it is ridiculously little effective in helping users enjoy their image collection. In particular, because it sticks to the underlying file system structure (no real-life user cares about this), then these browsers can be of limited use as soon as we forget *where* our images are located.

Assigning a unique location for a particular image can be very counter-intuitive. Where should one store a particular photo? The choices may include: in a folder associated with the time the image was taken; in a folder associated with the event captured by the image; or in a folder associated with a key person in the image. In essence, there is no *logical* reason to prevent files being found in multiple locations. Instead, the alignment of current browsers with the underlying file system structure has many *practical* reasons, with simplicity perhaps the key reason.

The search capabilities of browsers, based on file names and/or file contents, can help in some limited cases. Furthermore, tags can be attached to images and used to find documents. Unfortunately, however, tags are treated very poorly as there are no means for structuring sets of tags in any meaningful way, e.g., by grouping tags into different concepts or by defining relationships between tags.

Addressing these issues is possible when taking an MDA perspective. Applying the MDA model to images offers opportunities to gain in terms of expressiveness; since media files can now be organized according to their logical relationships instead of being physically stuck in one place, it should be much easier for users to locate the set of media files of interest than it is with current browsers. Gains are also expected in terms of user interface; by browsing media files according to multiple visual dimensions, appropriately rendered on screen, deeper understanding of the media collection should be greatly facilitated.

Based on the resounding success of OLAP applications, it is not surprising that multimedia researchers have studied the application of OLAP to multimedia retrieval for some time (e.g., see [Zaïane et al., 1998, Jin et al., 2010, Arigon et al., 2007, Worring and Koelma, 2013]). The fact that the MDA model is geared towards simple numerical attributes, however, is a serious limitation when it comes to multimedia collections where tags and annotations are a very important part of the meta-data.

The use of tags has been studied in faceted search, however (e.g., see [Diao et al., 2010, Yee et al., 2003, Bartolini and Ciaccia, 2009]. Faceted search uses a single tag-set, but proposes to build multiple hierarchies (or even DAGs) over that tag-set, one for each aspect that could be browsed. These hierarchies are then traversed to interactively narrow the result set, until the user is happy. Item counts or sample queries are typically used to present the result when it is very large; when it is sufficiently small it is presented in a linear fashion. A major drawback of the faceted approach is the use of a single tag-set; although the hierarchies do help users somewhat to disambiguate the different uses of an ambiguous tag, it is more logical to categorize the tags into different tag-sets.

The ObjectCube Foundation

We thus started to take this MDA perspective on multimedia data modeling. We then designed ObjectCube, which is a new multi-dimensional model for media browsing (for details see [Tómasson, 2011, Tómasson et al., 2012a, Jónsson et al., 2014]).

The ObjectCube Data Model. The concepts of *Objects* and *Tags* are at the core of ObjectCube. An *object* is any entity that a user is interested in storing information about, e.g., an image file. A *tag* is any meta-data that can be associated with objects. Many objects can share a tag and many tags can be associated with a single object.

ObjectCube has also four additional concepts that concern categorization and grouping of data. ObjectCube defines *tag-sets*, which are used to organize tags. A tag-set is a (mathematical) set of tags that the user perceives to be related; we can think of the tag-set as a category of tags representing a concept. The 'People' tag-set naturally gathers names of people or names of subcategories, say, 'Children' or 'Class Mates'. Tag-sets may be manually created or automatically created thanks to contents analysis, e.g., face recognition for the 'People' tag-set.

It also defines the concept of *hierarchy* which is a tree that adds structure and order to a subset of the tags of a tag-set. More informally, it serves to categorize some of the tags of a tag-set. The 'People' tag-set, e.g., could have one hierarchy called 'Friends' and another called 'Family', possibly disjoint, possibly sharing tags.

Finally, ObjectCube defines the concepts of *hypercube* and *cell*. A hypercube is created by selecting and storing information about one or more tag-sets, or hierarchies, which the user wishes to browse objects by. A cell in the hypercube is the intersection of a single tag from each of the dimensions (tag-sets or hierarchies) in a hypercube.

The above concepts offer users flexible ways to model their data which are subsequently retrieved from a database at browsing time via a series of *filters*. A filter is a constraint describing a sub-set of objects that the user wished to retrieve. Each filter applies to a single tag-set, but many filters may be applied to the same tag-set. A filter can be applied to any dimension, regardless of whether that dimension is visible in the user interface or not. Filters can select tags, can define ranges of interest or can be applied to hierarchies to select specific nodes (and their sub-hierarchies). Note that tags, tag-sets, and filters are strongly typed, for both conceptual and performance reasons.

A Prototype. We have already developed a prototype media server (see [Sigurþórsson, 2011, Tómasson et al., 2011, Tómasson et al., 2012b]) implementing the ObjectCube model, mainly to check whether the data model we propose can be efficiently implemented. The prototype has a central logic module implementing the data model, as well as APIs for data storage, user interface development, and automated media analysis using a *plug-in* architecture.

Plug-ins are code libraries called upon to analyze media objects during insertion and generate new tags or attach objects to existing tags. Currently, three plug-ins are implemented that: extract the EXIF meta-data associated with media files; extract faces from photos [Rúnarsson, 2011]; and analyze the color composition of photos.

We evaluated its performance in a realistic context, using three different underlying datastores and image collections containing up to one million photos. We ran extensive performance evaluations checking the scalability of simple tag filtering, range filtering as well as hierarchical filtering. We measured the response time of the server when retrieving data sets having varying cardinalities, and this for three off-the-shelf persistent storage managers. It basically shows that column-store oriented data storage systems best cope with scale and that the ObjectCube is indeed usable in practice as it returns answers very rapidly. Detailed performance results are reported in [Tómasson et al., 2012a].

VI.2.5. Research Directions

Enabling multidimensional media browsing asks to work along at least three research directions. Initial thoughts along each direction already gives a dense research program which is outlined below.

Research Direction #1: Data Model, Concepts and Implementation

Handling Similarity. So far, we had no blocking difficulty to turn the pure and traditional OLAP model into the ObjectCube conceptual model that is more multimedia oriented. We have already defined how to map some of the OLAP concepts to ObjectCube, as well as we have taken great care in making sure ObjectCube can be efficiently implemented. It is unlikely, however, that the ObjectCube is today complete enough to adequately model all aspects of media browsing.

Traditional media search involves presenting a query to the system (e.g., key words or an example of a media file), finding media items considered similar to the query, ranking the media items according to some similarity, and presenting the ranked list to users, sometimes supplemented with a score.

Similarity search does not have a direct counter-part in traditional OLAP applications. It is so far unclear whether supporting the similarity between data items requires a new concept in ObjectCube or if similarity can simply be constructed by assembling the existing concepts.

Handling the Temporal Dimension of Multimedia. It is compulsory to integrate videos in image collections, as they are de facto a highly popular media. They key difference with what we are starting with is the temporal dimension of these documents. Of course, OLAP copes with time to a certain extent, as facts can be associated to dates, but this is fundamentally different from the time that is an internal dimension of documents, as when considering videos. Adding a time dimension to an OLAP model revisited for multimedia is very challenging. It requires to make sure the model itself remains valid from this temporal perspective; it also requires to make sure interfaces can appropriately distinguish between time between videos and time within videos. Furthermore, tags may apply to particular time-frames, as well as bounding-boxes, and topics such as object/person-tracking become relevant. It is not clear what concepts should be invented to provide a nice and clean notion of time in a multimedia oriented MDA data model.

Aggregating Data. The traditional OLAP model has been successful in part because it defines the notions of (pre-computed) aggregates that allow information to be analyzed at different levels of details. OLAP aggregation operations are "simple" as they often boil down to summations, averagings, etc. Aggregating multimedia documents is far from being that simple. What does it means to aggregate hundreds of pictures? What does it means to aggregate the many faces automatically discovered in images? What data should be used to build aggregates, and when should they be built? What conceptual properties multimedia aggregates should have, and how should these properties be related to the base data? Understanding how to map the simple OLAP aggregation operations to their counterpart in ObjectCube is one of the difficult questions to tackle.

Indexing and Scalability Issues. Traditional OLAP applications come with their own set of indices and access structures. This is making sure the OLAP model can be implemented efficiently, otherwise it would have no practical interest. The same holds for its multimedia embodiment.

We need to study whether the traditional OLAP-oriented access structures are beneficial for multimedia or whether entirely new structures are needed. We need to study the performance of the current model in more detail, for hierarchies with very high fan-out, and for different levels in hierarchies. We must understand how this relates to (approximate) high-dimensional indexing techniques. It is highly likely that new methods of access must be created for adequate performance.

Research Direction #2: Extracting Contents from Media

The MDA model needs to be fed with data. That data comes from users taking pictures and from data automatically extracted by various image analysis tools. Overall, this allows to structure the data according to the data model, where data items are given their appropriate logical location as well as their relationships according to hierarchies, tags, etc.

Information Flow. It is therefore key to understand how to connect media analysis tools to the data model in order to populate it with meta-data. This raises many issues. First of all, it raises the problem of understanding how do the multimedia data and the meta-data are flowing between the various components. There is an obvious flow of meta-data from the tools extracting contents in order to populate the data model. There are also more complex flows of data between the (many) tools that need to learn form a collection. Furthermore, a personal photo collection is dynamic, hence, how does this impact the meta-data already computed? Overall, regardless of these examples, it is key to understand how information is flowing between the various conceptual components when modeling multimedia data.

Coping with Errors. Relying on automatic meta-data extraction also raises the complex problem of dealing with errors. Existing tools are pretty good at detecting and extracting faces for example, but they often make (sometimes hilarious) errors. We all know that it is extremely painful to correct every single error, so how can we correct errors, how can we feed learning algorithms with better examples in a way that is not that painful?

Ubiquity. Personal photo collections are likely to be browsed from multiple places, on multiple devices. It can for example range from a desktop connected to a large screen to a cellular phone. Overall, multiplying devices all with different resolutions, different computing power, different mobility requirements is asking for adapting the way things are displayed in a clever manner. This is far from being straightforward and calls for involving tools such as the automatic selection of region of interest, visual attention models, repurposing algorithms. All those tools are today making the bulk of the state-of-art contributions in multimedia conferences every year. Overall, this is suggesting that we have to make sure the concepts underneath such techniques can fit with the MDA model. Note that we believe that the restricted perspective of photo browsing (that contrasts with multimedia in general) offers opportunities for an improved extraction of meta-data from photos as the domain is more narrow, likely limited to families and friends.

Machine Learning. On a more conceptual level, such tools often rely on similarity metrics, on quantization, on sparse representation models, on kernel functions, on machine learning, on ranking, on projections, on dimensionality reduction strategies, on Bayes, etc, and none of these concepts have their counterpart in OLAP. There is therefore a strong effort to make in order to make the OLAP and machine learning for multimedia worlds as compatible as possible.

Research Direction #3: GUI, Usability

We foresee several directions here. User studies are needed to make sure the system is usable in practice by real users. Scalability of presentation is an important topic, as displaying photos on a screen requires particular visual representation that are clear, easy to understand, while showing as much and as useful information as possible.

It is pointless to develop a system that is too complicated for having any practical use. It is absolutely key to involve in our project people that have a solid expertise in interface design and user studies. We did such an initial study that was only good enough to show us that we were far from knowing how to proceed [Sigurþórsson, 2011]. Doing rigorous user studies is key for acceptance. Furthermore, however, we also need to include users in the design process of the interface, in order to better understand their priorities and habits.

When a media collection is viewed at a high-level, it is often more important to detect pattern and trends, rather than observing individual items. The question is then to know how to show such patterns and trends for users to make sense of their data. This is of course connected to the problem of aggregating data mentioned earlier. There are many visual tricks proposed in the literature for viewing large numerical data sets, and it is unclear whether these approaches fit with photo browsing. Furthermore, it might be useful to browse data according to a three-dimensional perspective that could be very informative for the user.

The three-dimensional representation of images and the structure of the hierarchies make the data model a suitable candidate for speech and gesture interfaces. Interface operations typically boil down to simple actions in the data model, and the mapping appears relatively straightforward. But going from traditional mouse-and-keyboard interfaces to more advanced interfaces is far from easy, and requires extensive work.

Other (Future) Research Directions

There are many directions that are very interesting and promising, but that we feel must wait until we have gained insights into the issues described above. These include:

Social networks. While the starting point of this work is single-user personal photo collections, we are well aware that sharing images with family and friends is central to any practical real world usage. Sharing is today mainstream with social networks, and, therefore, it is key to eventually clarify how these modern user-oriented usages can be linked to our proposal. Some questions already emerge: how can the data model support sharing of media, in terms of accessibility, privacy, performance, usability and other metrics; how hierarchies and meta-data can be reconciled between users sharing events tagged differently, yet consistent from a semantic point of view, ...

Cloud. This discussion is typically assuming the personal photo collections are stored on the local drives of users. While this is likely to facilitate an initial understanding of what is at stake when pushing the OLAP model to multimedia, it is an unsatisfactory assumption. It is likely the cloud will become a premium storage service and data will be outsourced, allowing easy access from anywhere, any time, reliably. Furthermore, in addition to relying on third party storage, it is likely services will also exist in the cloud, turning image/multimedia browsers into applications running in the cloud. Overall, this raises several challenging issues that are somehow related to the ones mentioned above for social network sharing. Understanding how

a third party outsourced service can work with outsourced data without jeopardizing privacy is central.

With outsourcing, it is key to understand the impact of hosting the raw data and/or the meta-data at the client and/or at the servers, as well as determining the best place(s) for running computations given costs constraints. This overall raises architectural trade-offs to study. Regardless of these architectural problems, this raises encryption problems as well as data integrity and completeness issues. Note that some of these issues have been already addressed in the relational database community, but there is so far no clear understanding of what happens when applying these techniques to multimedia data. Note also that we come back on privacy issues in the next section.

MOOC. The last decade or so has seen a tremendous change in the way people use and enjoy computers. Young people today are growing up in a social-media-driven, everything-is-connected kind of world, where information is a their fingertips. Such changes also impact education. Many universities have started offering so-called Massive Open Online Courses, or MOOCs, where anyone can register for a fee and study. These courses are very popular, but the learning environment is difficult and dropout rates are very high. Online courses typically aggregate multimedia material, such as a video of lectures, possibly with slides.

One possible reason explaining the little success of the current MOOC initiatives is the crudeness of the material that is made available to trainees. Most lectures are put online in their raw form, with little or no means for browsing the contents in a flexible way. There is no mechanism to easily skip a part, to jump to the next key concept detailed in the course, to adapt the contents to the level of knowledge of the trainees, etc.

The vast majority of such material remains unprocessed and (hour) long videos are the only things student can work with. Needless to say such raw material is not appealing and can in part explain the poor success of electronic courses. In contrast, some lecturers have taken special care to facilitate the takeup of such material by students. In this case, lectures are typically chopped into (short) knowledge units, table of contents and/or an index is superimposed over the taught topics, and handouts exist as well as some other material such as links to text books, exercises and other related resources. This, however, is the result of a long editing process which finally provides a very flexible learning context with navigation facilities. Such flexibility should be the de facto standard for online training but the daunting manual burden is a severe blocking factor. This hand-crafted flexibility is so far an entirely manual task, which naturally limits the number of "enriched" courses teachers can propose online.

Recent advances in multimedia research is likely to facilitate this enrichment with contributions (in part at least) automatically processing multimedia streams to uncover segments, thread similar topics, extract keywords, see for example [Petrushin and Khan, 2007, Merkt et al., 2011, Bhatt and Kankanhalli, 2011, Kay, 2012]. Some research in multimedia is trying to automatically analyze and describe video and textual contents. They propose strategies for synchronizing all material available across modalities [Snoek and Worring, 2005], creating and characterizing links between related material [Law-To et al., 2010]. This obviously applies to online courses. Weaving links between taught material, chopping a long video into segments that have some thematic consistency [Van Mulbregt et al., 1998, Poulisse et al., 2014], allowing the browsing of segments, navigating between them [Eskevich et al., 2013], possibly searching for keywords, taught concepts, etc. is a hot topic in the multimedia community.

So far, researchers tackling these problems (see [Zhao et al., 2013]) need tools such as high-dimensional indexing techniques to appropriately handle the low level features representing multimedia elements. They also need pattern recognition/discovery approaches to detect motifs that

are repeated within the streams [Yuan et al., 2012, Chen et al., 2011, Ta and Gravier, 2012]. Additional approaches are needed to extract as much semantics as possible from the contents, thus including ASR and NLP contributions.

Overall, some research in multimedia goes that route, trying to segment multimedia material in meaningful units, with links, etc. However, there is very little work on any data model for subsequently navigating within the collection of multimedia bits and pieces. Applying the MDA model to this MOOC settings is part of future work.

VI.3. Securvacy (Security+Privacy) of Multimedia Contents

For many years, we have worked while having in mind the goal of constructing content-based retrieval system working at large scale. To some extent, this goal has been reached, by us, as well as by other research teams elsewhere. Publications in good venues detailing experiments prove that the designed systems do work. Systems are evaluated against large benchmarks, which is overall showing they are robust and scalable. Robustness is a consequence of using extremely powerful feature extraction schemes for doing image recognition, such as the SIFT image description technique. Scalability is a consequence of using approximate, hence fast, search techniques; approximations are such that they do not ruin the good recognition power of the descriptors. While there are still many research problems to solve, it can be said that content-based retrieval systems are now quite mature, and, as a result, there are several systems that are used out in the field, in real life.

Without any loss of generality, content-based multimedia processing have so far been used in very friendly settings where the enrichment of knowledge is paramount. This typically benefits to content providers, increases users digital experience enjoyment, etc., sometimes with economic consequences. For example, there are some neat applications such as Shazam which does song recognition using low level audio signatures and content-based similarity searches. It copes with strong background noise, as it might be in bars or other public places. Subsequently on-line buying the recognized tunes is made extra easy. Other content-based systems do recommendation [Aghasaryan et al., 2013] based on the contents of multimedia items in addition to relying on the classical collaborative filtering schemes. Furthermore, grouping users having similar profiles often boils down to running high-dimensional k-nn searches. In the same spirit, monetizing contents, proposing specific advertisements is possible thanks to content-based approaches.

Processing multimedia contents now also happens in more hostile settings where the control, the surveillance and the filtering of multimedia information are central. In this case, systems no longer magnify any cultural richness, but protect the commercial value of contents. This is typically the case when content-based retrieval systems are used to detect copyright violations, filtering multimedia contents in order to protect the creation of the few from the piracy of the many [Lejsek et al., 2005, Lejsek et al., 2006b, Law-To et al., 2007]. For instance, content-based retrieval systems can spot the upload of copyrighted material on sharing platforms (YouTube) to either block or monetize it (i.e., advertising revenues are shared with the copyright holders). Automatically analyzing contents can also be used to feed a parental control system in order to hide improper material. In the same spirit, there is also an increasing number of forensics applications that rely on analyzing the contents of multimedia material: we have been mentioning the automatic detection of illegal copies of copyrighted material, but there are many other sensitive contexts such as the detection of child pornography, of terrorist images and video propaganda. Analyzing contents not only helps blocking this material, but also fine grain processing it might help dismantling networks.

Because there are valuable goods to protect, because there is some very sensitive information to steal, serious pirates might try to attack content-based retrieval systems. An attacker might want to by-pass a parental control mechanism, might want to evade a filtering system and indeed upload some copyrighted material, might want to bias the results of a search that is used as a foundation for monetizing contents. An attacker might also want to spy on the queries sent to a CBIR system by users in order to infer their preferences, to know what are the images they are looking for, etc.

This section is questioning the use of multimedia content-based retrieval system in adversarial environments. The bulk of the text, however, focuses on still images with which we have mostly been working for many years. Yet, what really matter are not the images *per se*, but the fact that we are dealing with high-dimensional features. Therefore, most of the discussions below apply quite well to other media such as videos, audio, etc.

This section is structured as follows. We first discuss what is challenging the privacy and the security of systems that are using images and running similarity searches. That review is traversing three research fields, Forensics and Counter-Forensics, Biometry and CBIR. We then move to proposing several *Securvacy* research directions we definitively want to tackle in the future, *Securvacy* being a word we coined to refer to challenging the security and/or the privacy dimensions of CBIR systems.

The discussions in this section have found their origin in a personal concern that comes on top of a scientific interrogation. The NV-Tree is the spine-bone of the *Videntifier Technologies* company which scouts illegal multimedia material. I would feel insulted if I discover that sick people can attack the system and break its great recognition power and its superb efficiency, and then continue with their dirty business. Then, thanks to Patrick Gros, Teddy Furon and Ewa Kijak, this topic gained interest here and serious work could start. The initial investigations were possible thanks to the Ph.D. work of Thanh-Toan Do. Deeper and broader investigations addressing the security and the privacy dimensions of CBIR systems started thanks to collaborating with Armen Aghasaryan, David Gross-Amblard, Hervé Jégou, Stéphane Marchand-Maillet, Arthur Masson, Benjamin Mathon, Paul Temple and Li Weng, in addition to everyday hard thinking and working with Ewa and Teddy.

VI.3.1. Starting Points

What is at stake in this section is the security and the privacy of multimedia material, typically images. The spectrum of applications that use images and assess their similarities as their foundational base is extremely large and goes well beyond the Computer Vision, Databases and High-Dimensional Indexing domains that were, so far, at the core of this manuscript.

The Forensics and Counter-Forensics domain, as well as the field of Biometry, root their applications in evaluating the similarities of multimedia items (essentially images, however). Furthermore, these two domains are centrally concerned with security and privacy. It is therefore inspiring to review the main challenges that were identified by researchers in those domains, and that are threatening the security and the privacy of these techniques. It is also inspiring to review the main contributions designed in reaction to these challenges, and that aim at increasing the levels of security and privacy of these techniques.

That review is not technical, it instead highlights the motivations and goals when attacking systems. It also highlights lines of defense when applicable. Overall, the goal of this review is to show the great diversity of security and privacy challenges researchers have identified in their respective domains, most of these challenges being applicable to the domain of multimedia retrieval and CBIR systems.

This review is structured as follows. It first discusses the domain of Forensics and Counter-Forensics, and then moves to the domain of Biometrics. For each domain, the discussion starts by reviewing motivations and goals, and cites key publications. Then, each discussion moves to a slightly higher perspective and comments the major challenges found in these publications. Finally, each discussion transposes these challenges to the context of CBIR systems.

Inspired by Forensics and Counter-Forensics

In the Forensics and Counter-Forensics literature, it is typically the authenticity of images that is the central point of interest [Gloe et al., 2007]. The topic of digital image forensics can be broadly divided into two main series of problems. The first series of problems is related to assessing whether or not a particular digital image has undergone *tampering*, i.e., malicious post-processing. Removing people or objects from a picture for the purpose of camouflage is a typical image manipulation; it is often performed by a copy-move forgery operation where a innocuous region of an image is copied and moved over the sensitive part [Christlein et al., 2012]. The Forensics literature therefore includes many algorithms designed to unveil the specific traces of such post-processing steps.

The second series of problems is related to identifying the source of the image, that is, the technology and the specific on-board device that were used to acquire the picture in the camera. Forging a fake origin or suppressing the traces allowing to unveil the origin of an image are typical manipulations from this second series of problems. The Forensics literature therefore includes algorithms identifying the statistical noise hidden in the images, from which identifying the camera, the CCD sensor, etc., is attempted.

Of course, in response to the contributions that can be found in the Forensics literature, a large body of Counter-Forensics approaches has been published. Counter-Forensics is concerned with smart adversaries trying to induce a certain outcome by impeding or misleading the forensic analyses of digital images [Bohme and Kirchner, 2013]. Therefore, Counter-Forensics approaches are on the one hand trying to exploit the weaknesses of forensics approaches, and on the other hand they are trying to improve the robustness and the reliability of the image forensics approaches.

Adversaries typically acquire a very deep knowledge about the models, techniques, implementation details of the latest forensics approaches. They then use that very specific knowledge to attack a particular forensics technique in order to cover their traces and/or to forge misleading traces. The design of many early forensics techniques was not taking into account the existence of such malicious adversaries. These techniques were good to assess the authenticity of images in contexts where tampering was not specifically designed to deceive a particular forensics technique; contexts where these techniques proved to be helpful were contexts where the tampering was "blind", "universal", very generic, using any photo editor standard tool and a fair amount of skills. This contrasts with "informed" tampering techniques that precisely know what the forensics analysis processes are doing. Informed techniques thus apply sophisticated image modifications specifically designed to pass through the gaps.

Recently, several approaches have extensively attacked the forensics processes that are relying on the use of the powerful SIFT descriptors. By precisely knowing the way SIFT keypoints and descriptors are computed over images, by precisely knowing how they are used to identify duplicated regions in images, very specific tampering techniques were designed. These techniques proved to successfully deceive the best forensics techniques based on SIFT [Caldelli et al., 2012, Amerini et al., 2013a, Amerini et al., 2013b].

Three major comments are in order before moving to the field of Biometry.

Comment #1: Robustness is Not Security. The Forensics and Counter-Forensics literature give us good examples of the split between techniques that are *robust* and techniques that are *secure*. Robust techniques typically gracefully handle generic tampering, such as copy-move, resampling, smoothing, compression, etc., done possibly with great skills and care, using some photo editor tool. But these techniques fail at identifying tampering when the offensive

manipulations are based on a deep knowledge of the internals of a specific forensics system. In contrast, secure forensics techniques of course detect generic manipulations. But, in addition, their design makes it extremely unlikely an attacker can successfully craft a specific manipulation that would remain undetected even if this attacker knows about the algorithms, the parameters, etc., used in this secure technique.

Comment #2: Impacting Results with False-Positives and False-Negatives. Overall, the outcome of the attacks challenging forensics approaches can be twofold. On the one hand, attackers try to cover their traces. In this case, the system is unable to identify that one image has been tampered while it has been. In terms of quality evaluation, this is a *false-negative* (FN). On the other hand, attackers might not want to cover their traces, but might instead trigger an erroneous identification. They can for example forge a fake CDD device signature or they can tamper the image such that this surrogate forgery creates a diversion making more problematic the identification of the truly relevant forgery. In terms of quality evaluation, this is a *false-positive* (FP).

Comment #3: Obscurity is Not an Option. The Forensics and Counter-Forensics literature as well as the Watermarking and Cryptography literatures show that obscurity is not an option to enforce security. Security through obscurity relies on the fact that attackers may have very hard time to precisely know what are the algorithms used in the system, what are the implementation details, what are the parameters and what can be their values. In other words, security of systems through obscurity assumes the attackers are unlikely to find the security flaws due to the great complexity of the system they are attacking.

It has been demonstrated that trusting obscurity is not reliable. Attackers look for any piece of information in publications, patents, standards or by social engineering. It is impossible to empirically assess how difficult it is to disclose information about a system.

The second drawback of security through obscurity is the high cost of changing the algorithm if it gets disclosed. Designers need to re-implement another obscure algorithm, likely way different, with heavy testing phases and a high burden for deploying the algorithm on sites. Furthermore, this takes time during which the system remains insecure.

These problems have been reported since a long time and Kerckhoffs came up with several design principles still applicable today [Kerckhoffs, 1883]. His best known principle says that the system must remain secure even if everything about the system, except a secret key, is public knowledge. A secret key is a piece of information that determines the output of a particular algorithm: the same algorithm does not produces the same output if it is fed with the same data but with different keys. This guarantees the security of the system.

As it is much easier to protect a small piece of information (the secret key) than a complete system by obscurity, secure systems using secret keys are much more reliable. A secret key is one or more very large random number. Finding its value by an exhaustive search is almost impossible as the key space is very large. If the key is discovered, then forging another key is easy and fast. Real-world secure systems typically include secret keys and elements of obscurity.

Inspired by Biometrics

Biometric systems typically rely on k-nn searches to identify the templates in the database that are the most similar to the one probing the system. This is the core of all CBIR techniques. It is therefore very relevant to observe the solutions enforcing the security and the privacy of k-nn searches and that are proposed in the Biometrics field.

The field of Biometrics is also making heavy use of images for the purpose of identifying people [Jain et al., 2004, Park and Jain, 2010]. Biometrics include considering one or more human physical characteristics for authentication, like the fingerprint minutiaes, face, hand geometry, voice and iris [Rathgeb and Uhl, 2011, Bowyer et al., 2013, Rane et al., 2013, Abaza et al., 2013]. These characteristics are typically captured by an image that is then probing a computer-based biometrics system. Biometric identifiers are unique to individuals. Therefore, they are more reliable in verifying identity than any other traditional system based on passwords and/or on personal identification numbers (e.g., the PIN code used when paying with a card).

Obviously, the robustness of biometric systems is extremely important. An individual should be identified even if his voice is altered by a cold, even if his fingerprints are modified by dirt or injuries, etc. The security of biometric systems is also important. Here again, an attacker may want to bypass a biometric system by specifically forging a fake human characteristics that the system may not be able to differentiate from a real one. Not being recognized and/or being recognized as being someone else might be goals attackers pursue.

Of all biometrics-based techniques, systems relying on analyzing the patterns in human irises proved to be very reliable and highly accurate in verifying the identity of individuals. Yet, their security has severely been challenged, see [Venugopalan and Savvides, 2011]. In this paper, Venugopalan and Savvides can generate alternate iris textures by reverse engineering the iris bit code of one individual that has been stolen from the database. They then embed this forged texture within a person's natural iris texture to successfully spoof another person's iris. Such attacks generate *false-positive*. Tricking the biometric indicators to generate *false-negatives* is another possibility, for example in surveillance-based systems where one individual wants to remain undetected despite being in the database. Recently, a more theoretical approach compared blind and informed attacks against authentication and identification systems that are based on content fingerprints [Beekhof et al., 2012]. It shows that an attacker who strives to counterfeit a particular item can decrease the performance of practical authentication systems, both in terms of false acceptance (i.e., *false-positives*) and false rejections (i.e., *false-negatives*).

Collections of biometric identifiers raise very serious privacy concerns, obviously, as stealing biometric data can endanger identification, as demonstrated by Venugopalan and Savvides, but can also eventually reveal the identity of the persons enrolled in the system as demonstrated in [Jain and Nandakumar, 2012]. For example, Adler in [Adler, 2004, Adler, 2003] can recreate, from stolen face template collections, artificial images of faces, endangering privacy. Note that the recreated faces are quality-wise good enough to fool the recognition algorithms.

Biometric systems thus rarely keep their data in the clear, fearing a pirate could steal such highly valuable data. The main axiomatic in biometric claims that no database can be stored securely. Similarly, a user is reluctant in sending his biometric template in the clear. Biometric systems therefore often rely on cryptographic primitives such as homomorphic encryption, oblivious transfer, argument based encryption, secure multiparty computation protocols. Such primitives are intended to protect the database of templates as well as to make very hard the forgery of fake templates. Thanks to cryptography, the database and the feature extraction process are thus more secure [Hsu et al., 2009, Lu et al., 2009, Hsu et al., 2010, Bringer et al., 2013, Erkin et al., 2009, Sadeghi et al., 2010, Osadchy et al., 2010].

Several comments are in order before moving to the field of CBIR.

Comment #4: Biometry Uses a Two-Way Privacy Model. Biometric applications typically adhere to the two-way privacy model. With this model, the user and the system (often termed the *client* and the *server* in the privacy literature) both wish to keep their data private, that is,

the system should learn nothing about the query from a user and the user should learn nothing about the contents of the database managed by the system, except for the results of the query.

Many contributions in biometrics research adopting the two-way privacy settings have been published, see [Barni et al., 2010, Erkin et al., 2009, Huang et al., 2011b, Sadeghi et al., 2010].

Comment #5: Cryptography Comes with a High Cost. Biometry has extremely strong privacy and security demands, and fully implementing the two-way privacy model is possible thanks to cryptography. With cryptography, running secure and privacy preserving k-nn searches typically relies on the notion of *Signal Processing in Encrypted Domain—SPED* [Barni, 2006, Erkin et al., 2007, Lagendijk et al., 2013, Rane and Boufounos, 2013, Barni et al., 2013]. Despite providing an excellent level of protection for data, the implementations of these protocols are extremely complicated, hence very costly. In practice, the computation of the Euclidean distance in the encrypted domain is slow and exchanges back and forth ciphers that are bigger in size than the high-dimensional vectors they represent. The search itself is exhaustive, running as many times SPED protocols as there are items in the database.

VI.3.2. Security and Privacy for CBIR Systems

CBIR systems share a lot of issues with Forensics and Counter-Forensics as well as with Biometrics. They typically all use images that are at the root of everything; they share many algorithms as well as methodologies; they also share some of the security and privacy challenges that were highlighted above as many of them directly apply to CBIR systems. There are, however, specific securvacy challenges that concern CBIR. The goal of this section is to discuss how the challenges highlighted above transpose to the context of CBIR systems. This section is also discussing other challenges that are quite specific to CBIR systems. We first start with security issues before moving to issues related to privacy.

CBIR Systems are Robust. Are They Secure?

The strength, the power of CBIR systems is typically evaluated via benchmarking their capacity to match images despite distortions creating quasi-copies used as queries. Distortions are typically crops, rescalings, rotations, occlusions, etc. Distortions are made in a very generic manner, by applying a set of transformations that are completely independent from the systems later performing recognition tasks. Overall, the more robust a CBIR system is, the more it will succeed matching distorted quasi-copies queries with their original, non-distorted counterpart database images; the more severe the distortions it can absorb are. This is typically what the TRECVID competition is all about, for which many research teams worldwide compete [Over et al., 2011], hoping to be best ranked. This is also how we evaluated the various high-dimensional indexing techniques we ourselves designed and described in this manuscript.

The Forensics, Counter-Forensics and the Biometry domains suggest systems have to be more than robust, as their security can be challenged by malicious attackers. This also applies to the domain of CBIR. Challenging the security of a CBIR system can for example mean a malicious attacker devises offensive strategies applying specific visual distortions to quasi-copies such that the system fails at identifying their original counterparts.

So far, CBIR systems proved to be robust but very few contributions discuss their security. With CBIR systems, security challenges recognition as well as more sophisticated tasks such as classification. Both are briefly discussed now.

Challenging Recognition

As it is the case for all forensics techniques, an attacker challenging a CBIR system may specifically modify an image such that the recognition power of a system is deceived, failing to recognize some contents that should normally be identified and/or forcing a "spurious" recognition that should normally not have happened. Overall, this translates into having a CBIR system producing *false-positives* and/or *false-negative* results.

Few papers have already investigated challenging the security of CBIR systems. The first paper we are aware of is [Hsu et al., 2009]. This paper shows that very specific anti-SIFT attacks can jeopardize the detection of keypoints, which in turn is making it hard for a CBIR systems to subsequently match the attacked images with the ones from the database. While Hsu et al. motivations and worries are perfectly correct and legitimate (and latter extended, see [Lu and Hsu, 2012]), and in some sense confirmed our desire to investigate that topic, their experimental protocol had some flaws. We analyzed in [Do et al., 2010c] these flaws and then subsequently designed anti-SIFT attacks producing *false-positive* and/or *false-negative*, see [Do, 2012, Do et al., 2010a, Do et al., 2010b, Do et al., 2012a, Do et al., 2012b]. Our most sophisticated attack relies on a dedicated picture-in-picture scheme able to delude the recognition capabilities of CBIR system that uses state of the art components. Running experiments using 100,000 real-world images show that in almost all cases, it is possible to produce enough *false-positives* to lower the trust a user can have in the system, making it useless in practical settings.

Challenging Classification

Recognition is not the only CBIR-task researchers have challenged from a security point of view. For example, Melloni et al. attacked the classification of images [Melloni et al., 2013]. The task is very close to recognition, yet, the ultimate goal is to classify images into several categories and to possibly automatically assign (or propagate) tags to images. Here, they show that a malicious user can for example modify an image in order to alter the tags automatically suggested by the classifier, and/or to have this image associated to the wrong class. To achieve this, they make an extensive malicious use of the internals weaknesses of the image description technique (SIFT), of the image indexing scheme (Bag of Features) and of the training phase associating tags to their annotated image regions counterparts. They show classification can be fooled without severely affecting the visual quality of the attacked image. This is a very serious threat as it might ruin the ability of a parental control system to detect and filter improper material [Yizhi et al., 2010].

It is extremely insightful to observe that in their work, Melloni et al. challenge the knowledge that is learned on a training set and that is used by almost all image classification strategies. In particular, they use a contribution from Liu and Wang who show it is possible to reverse engineer the internal mechanisms of an image classifier to reveal the so-called support regions that determine the outcome of the classifier [Liu and Wang, 2012]. By specifically altering the visual contents of these support regions, Melloni et al. deceive classification.

This is raising two issues. First, reverse engineering the internal mechanisms of an image classifier can appear to be a serious privacy threat. A malicious attacker doing this, using the technique presented in [Liu and Wang, 2012], can somehow *confiscate* a very valuable know-how, the foundations of the classification used in this system. This, in turn, may economically be harmful to the research lab/company having some business built on that classification mechanism. We will come back on such privacy threats below. Second, it raises the general issue of challenging the machine learning processes that are used in the context of multimedia and by many approaches.

Challenging Machine Learning

In general, multimedia systems contain a fair amount of machine learning techniques. These techniques typically need data for training. Then, what has been learned during this training phase is used by the system to process new data items. Classical machine learning techniques used in multimedia include (but are not limited to) PCA, SVD, SVM, GMM, decision trees, all nearest-neighbors based methods, kernel methods, hidden Markovian processes for modeling multimedia, CRF, etc., see [Chang, 2011].

To our knowledge, very few approaches have so far challenged the security and the privacy of machine learning processes that are used in multimedia systems. Liu and Wang with their publication (see [Liu and Wang, 2012]) is a remarkable exception. Interestingly, several papers are discussing the security of machine learning techniques in adversarial environments in the domain of Spam detection, see [Barreno et al., 2006, Laskov and Lippmann, 2010, Barreno et al., 2010, Huang et al., 2011a]. [Biggio et al., 2013] focuses on the problem of data clustering in adversarial contexts while [Biggio et al., 2012] shows how to poison SVMs in the context of classifying spam as well as classifying images of digits. This work is closer to the problems at stake in multimedia, yet, the images of digits are extremely simple and pixels are directly used to determine the outcome of the classification.

Threatening Privacy of CBIR Systems

Biometry gave us good examples of privacy threats, as well as technical solutions for protecting data. CBIR systems are also obviously concerned with privacy threats.

An attacker might want to know about the images that are stored in the database. He might also want to spy on the queries submitted by a user to then determine what this user is interesting in. Real life examples have already demonstrated such privacy violations with text queries, however, but it is easy to transpose them to multimedia: if someone repeatedly searches for tumor images, it might be possible to infer that this specific user is concerned with a particular medical problem. Note this problem is not fixed by uploading at the system the signatures instead of directly uploading the contents, as demonstrated by Weinzaepfel et al. [Weinzaepfel et al., 2011]. This paper demonstrates that images can be reconstructed from their SIFT description. If the attacker is spying more than one user, then it is possible to find correlations between what these users query or upload, severely endangering privacy.

Two-Way Privacy for CBIR

We discussed above the two-way privacy model that is used in all Biometrics applications. To meet the extremely strong privacy demands, cryptographic techniques enforcing this model are typically very heavy in computation/communication. Costs are typically linked to the size of the encrypted database. As an example, it takes about 10 seconds to run a secure k-nn search over a database containing 320 entries, see [Lagendijk et al., 2013]. With CBIR systems, databases include million to billion items. Scale thus prohibits using such cryptographic primitives.

Furthermore, there is almost a philosophical mismatch between cryptography and searches by similarity: cryptographic tools are typically exact and do not tolerate any alteration, any loss when representing/decoding ciphered items. In contrast, content-based systems and multimedia items must inherently cope with distortions; distortion at acquisition time, distortion at feature extraction time, distortion at quantization and indexing time, etc. It is because CBIR systems/data can cope with distortions that similarity searches are possible. A content-based system would have little practical use if exactness was the only way.

One-Way Privacy for CBIR

Recently, however, another privacy model has started to be investigated by researchers. This model is called the *one-way privacy model*, see [Fanti et al., 2013]. One-way privacy settings assume that only the user wants to keep secret the information he owns since the database managed by the system is public. Public databases against which users may wish to run private queries have become commonplace nowadays. Some public media databases already integrate similarity search mechanisms, Google Images or Google Goggles. It is likely others will soon follow that path, turning Flickr, YouTube, Facebook in content-based searchable collections (in addition to already being tag searchable).

Private Information Retrieval (PIR) techniques allow a client to retrieve a particular data item (using it's identifier e.g.) from a database without revealing the query to the server. Other PIR techniques use multiple servers to achieve information-theoretic security, such as the one by Chor et al., see [Chor et al., 1995]. But their communication complexity is order-equivalent to a full database transfer, which is problematic when its cardinality reaches billions. Furthermore, as for the SPED-related approaches, such techniques require *exact* searches, which somehow clashes with the notion of *similarity* that is central to multimedia searches. To address this specific issue, techniques have been proposed to run exact searches for a broader group of encrypted entries in the database, that group probably including the desired content. Then, some additional work is typically done at the client to select the correct item. Such techniques have been proposed by Shashank et al. ([Shashank et al., 2008]).

We recently proposed one slightly different approach that is also targeting the enforcement of a one-way privacy model. In [Weng et al., 2014], Weng et al. propose to omit certain bits when transmitting the feature describing the contents of the query. The reduced information sent to the server makes the query ambiguous—when n bits are purposely omitted from the query feature, the server runs 2^n consecutive searches, filling the missing bits. By construction, the true value of the feature remains unknown but will for sure be used at the server when probing its database. The server then returns all possible candidates to the client, the client then filters the candidate list to find the best match. With this solution, no encryption is used, yet, it is very hard and computationally difficult for the server to exactly know the user's interest.

Overall, the recent techniques contributing to the field of one-way privacy show it is possible to marry scale and privacy.

VI.3.3. Securvacy: Research Directions

The above discussion clearly shows that the securvacy of CBIR systems, i.e., their security and the privacy dimensions, can be challenged in many ways. It is now time to review the research directions we are likely to follow in the future.

Research Direction #1: Clarifying Trust Models

Trust models have extensively been studied in the computer network security and cryptography domains. Any security/privacy analysis relies on a trust model which lists the actors playing a role in the system and whether they can be trusted or not. In addition to actors, a trust model needs to list the possible targets of the attacks, the possible goals of attackers, as well as their level of knowledge. Furthermore, a trust model defines metrics for evaluating the success rate of the attacks.

We are very far from having on trust models for CBIR systems a view that is as clear and as precise as the one we have for the domains of computer network security and cryptography.

For that reason, it is mandatory to work on understanding how the threat analysis framework can be adapted to the context of CBIR systems. This section proposes initial thoughts in this direction, enumerating the main items of trust models.

Family. There are typically two families of attacks against a CBIR system when challenging its Privacy and/or its Security dimensions. The families are *Privacy attacks* and *Security attacks*. Attacks belonging to the Privacy family are trying to *learn something from a target*, while attacks from the Security family are trying to *act on a target*.

Actors. There are typically three different actors in any scenario challenging the security/privacy of a CBIR system.

1. The *Contents Owner* is the owner of the Works.

2. The *Contents Server* is where the CBIR systems is running. At the *Contents Server*, signatures are typically indexed in a database. Indexing has been performed by the *Contents Server*. Signatures (i.e. the *Meta-Data*, see below) may have been uploaded at the *Contents Server* by the *Contents Owner*, or directly computed there if the *Contents Server* received the contents (i.e. the *Data*, see below) from the *Contents Owner*.

3. The *User* who comes up with a particular image/video/audio to search for.

The trust model states which actors honestly play open cards and which actors want to delude the system. There are a priory many possible trust models as any of the three above-mentioned actors, or even worse, a collusion of several of them, can be suspected of dishonesty. The most classical scenario is when the user is attacking the system, all other actors being trusted. Another trust model is, for instance, a right-holder modifying his contents in order to increase the recognition rate because he receives incentives whenever the query is deemed to be a copy of his Works.

Note that it makes a lot of sense to investigate what is at stake when using an initial model where the attacker is curious but honest. In this case, the attacker is following the rules but is free to access all the information that leaks from its interactions with the system. We used this model in the context of one of our own contribution enforcing the privacy of content-based retrievals, see [Weng et al., 2014]. In contrast, we purposely adopted a malicious behavior when trying to delude the system using informed picture-in-picture attacks, see [Do et al., 2012a]. This later model makes things much harder.

Target. There are typically five targets that are:

1. *Data*

2. *Meta-Data*

3. *Contents Owner*

4. *User*

5. *Contents Server*

Data refers typically to images, videos, audio tracks. We have described in the sections above strategies to visually alter images such that the eventual recognition is made harder. Here, the attacks are done directly in the pixel domain for images.

Meta-Data typically refers to signatures such as the descriptors computed from the data, SIFT for images, MFCC for audio or any vector of movement for videos, e.g. It also refers to the information used by the internals of a CBIR system to index, quantize, classify, etc. Centroids determined after having ran a k-means on the meta data (the descriptors) extracted from the data (images), the parameters of a separating hyperplane, the probabilities attached to vertices as well as what edges connect within a hidden Markovian model, the frontiers of quantization bins are examples of meta-data one attacker might want to act on, or to learn.

By trying to learn from data or meta-data, or by trying to act on data or meta-data, an attacker can indirectly target attacking a *Contents Owner*, a *User* or a *Contents Server*.

Scope. Attacks span a continuous spectrum. At one end, an attack might aim a specific target or a small set of targets. In this case, we may say that this attack is *Focused*. At the other end, attacks might want to have a more indiscriminate impact. In this case, we may say that this attack is *Unfocused*.

Focused attacks might for example try to make sure a CBIR system fails at identifying a quasi-copy of a specific image, or might try to know what a specific user, or a small group of users do with the system. In contrast, unfocused attacks might for example try to degrade the overall recognition power of a CBIR system, might try to severely alter image classification in general or might try to steal as much information as possible from correlating users, queries and answers.

Goals. There are many goals for attackers challenging the securvacy of CBIR systems:

1. *Producing False Positives*
2. *Producing False Negatives*
3. *Acquiring Knowledge*

Producing False Positives/Negatives are two goals that we already highlighted when we discussed security challenges for Forensics, Counter-Forensics and Biometrics. They trivially exist in the domain of CBIR systems; we even provided few examples above.

Producing false positives deserves a special note. FP can be caused by a specific manipulation of the query, as the one we designed ourselves when running picture-in-picture attacks [Do et al., 2012a]. But the context of CBIR systems offers a unique possibility to attackers: polluting the database. The strategy Biggio et al. use in [Biggio et al., 2012] poisons the training set used by a SVM to degrade the quality of a Spam filter. This idea can be transposed to the case of CBIR systems. A malicious attacker can craft specific content that is inserted into the database to increase the success rate of a subsequent attack. For example, specific innocuous images containing redundant textures can be inserted in the database to later facilitate the picture-in-picture attack described in [Do et al., 2012a].

Acquiring Knowledge is typically challenging the privacy dimension of the system. Aside spying on queries, an attacker might want to learn about the system. By carefully crafting queries and checking answers, an attacker might determine the nature of the software blocks used in a particular system, stealing this know-how. In the same spirit, it might be possible to approximate the visual dictionary of a Bag Of Feature-based indexing technique, that dictionary being the corner stone of the spied system.

Knowledge. The level of knowledge an attacker has is of course crucial. Obscurity being a poor protection, enough implementation details will eventually be known to endanger a system.

On the other hand, very few CBIRS use a secret key. Some use the secret key to generate a private selection of parts of the contents [Kozat et al., 2004] or of parts of its features. Another approach defines a secret transform extracting some private features [Malkin and Venkatesan, 2004]. In their papers, Johnson and Ramchandran ([Johnson and Ramchandran, 2003]) and Swaminathan et al. ([Swaminathan et al., 2006]) define a secret quantifier used to quantize the extracted features.

Other attackers might not even try to estimate the secret key, which is likely to be impossible, but rather try to guess what in the data given to a system matters given the current secret key. For example, a secret key might drive the way colors get quantized at image signature creation time [Swaminathan et al., 2006]. Instead of estimating this key, which would in turn reveal the entire quantification process, it is possible to carefully play with pixel colors of attacked images, and check whether the system decides they are copies or not. By multiplying such tests, it may be possible to find the closest content to a given image which is not deemed as copy. Note that getting the secret key itself is never tried here.

Knowing the Contents of the Database. A critical point specific to CBIR systems security is whether or not an attacker (partially) knows the contents of the database. Rising false alarms becomes much easier if the attacker has this knowledge. It can simply be acquired thanks to the reputation of the image server making public the names of solid clients it is working with. Or, specific database probing protocols can be cooked by the attacker to get a glimpse of the contents. In the case of the one-way privacy model, the database is known. This might facilitate attacking systems.

Research Direction #2: Understanding Multimedia Outsourcing

Outsourcing multimedia material is what millions of users do every day: people share their personal images on Picasa, Flickr or Facebook. Outsourcing is very beneficial to many users. Outsourced multimedia repositories are typically available 24∗365, they nicely cope with failures, these two properties being costly to achieve for individuals. In addition, outsourced services might propose very interesting/useful features for users, such as automatic image classification, etc. For example, in 2010, Zhang et al. published a paper where personal images are classified according to an automatically inferred family tree [Zhang et al., 2010a]. This is very nice work, and such fun and handy services might be soon available as outsourced on-line add-ons.

However, it is well known that outsourcing data poses extremely severe security threats as third parties can actively seek out end-user information from multimedia contents. Our society has little choice: outsourced services for multimedia data are here today, and will become even more frequent in the near future as the general trend of cloud-computing fosters outsourcing. It is not acceptable to live in a world where our multimedia material can potentially be misused due to the inadequacy of current technologies.

Outsourcing is threatening the privacy dimension of CBIR systems. So far, most of the works dealing with multimedia items and outsourced services focus on privacy preserving personalization mechanisms for recommending items. Recommendation (for movies, URLs, musics, goods, etc.) is a very hot research topic, especially when merging the somehow traditional collaborative filtering techniques to more advanced content-based recommendation approaches. With recommendation systems, users are typically profiled, their profiles are sent to content/service providers which, in turn, send back personalized contents and/or services best matching their

centers of interest. Users are today forced to trust the content or service providers for the use they will make of their profiles, hoping no sudden disclosure. Users have no choice: they give this very sensitive information in order to benefit from personalized content offers.

Because this information is highly sensitive, several interesting privacy preserving schemes for recommendation have been proposed, see for example [Guha et al., 2011, Toubiana et al., 2010, Aghasaryan et al., 2013]. They all profile users locally before generalizing profile details into higher level categories of interest. Some form of anonymization is then applied and the resulting information is then sent to a server, eventually answering with recommendations.

These approaches are effective at preserving the privacy of users. They are however typically negatively impacting the quality of the recommendations as true personalization is more difficult to achieve due to generalization and anonymization. There is almost another philosophical mismatch between privacy requirements and the experience enrichment an individual can benefit from by using sophisticated outsourced services. It is as if someone is asking a system not to spy on images, but at the same time wishes face recognition and classification.

The database community has been puzzled since a long time with making sure outsourced services indeed do what they pretend to do [Ergün et al., 2000, Feigenbaum et al., 2002, Sion, 2005]. Overall, these approaches rely on strong cryptographic proof given by the outsourced service provider guaranteeing that the queries were actually executed correctly over their entire target data set. We are far from understanding how such mechanisms can be transposed to multimedia. The same applies to the concepts of k-anonymity and l-diversity quite well established for traditional database systems, see [Sweeney, 2002, Machanavajjhala et al., 2007].

Research Direction #3: Data/Meta-Data Authentication

Authentication is required in any context with outsourcing. The integrity of data (no data tampering), its correctness (answer truly exist in the database), its completeness (answers are built using all data, no more, no less) and its freshness (the results are based on the most up-to-date version of the database) must be guaranteed. An inspiring background comes from the relational databases security research where such problems have been tackled. Multimedia, however, asks to perform approximate similarity searches in high-dimensional spaces which are by essence very different from anything standard in secure database management systems.

Most of the traditional tools used to enforce data authentication, such as the ones based on Merckle Trees (see [Li et al., 2008] and [Martel et al., 2004]), can only deal with one-dimensional data. Very few schemes doing authentication of higher dimensional datasets have been published, see [Cheng et al., 2006, Tao and Papadias, 2004]. They are typically very expensive and face scaling problems. For example, the scheme proposed by Tao and Papadias is based on R-Trees, known to poorly work at scale with high-dimensional datasets.

We must invent new data structures that scale and that remain efficient when dealing with high-dimensional data to make authentication possible. Many high-dimensional scheme use some form of hashing, which is also at the root of the traditional Merckle Tree schemes. The proximity of these approaches from two very different domains seems a promising research direction.

Research Direction #4: Poisoning Multimedia Analysis

It has been demonstrated that it is possible to poison machine learning techniques in the context of the detection of Spam. Many algorithms used in the context of multimedia also rely on machine learning approaches, and, therefore, we might wonder about the effectiveness of poisoning multimedia analysis.

There are several crucial differences with respect to the assumptions made by Biggio and his colleagues (see [Biggio et al., 2013]). One difference is scale, where we deal with millions to billions items in multimedia while their studies assume orders of magnitude less data. A direct consequence of that enormous difference of scale is likely to impact the effectiveness of the poisoning: adding few poisoned data points to a learning set is enough to effectively impact the hyperplane separating Spam from Non-Spam—they deal with few hundreds to few thousands data items; in our case, many more poisoned items are likely to be needed before we see any influence on the outcome of the data analysis phase. But who knows? Outliers have a strong influence on PCA, clustering, classification, and carefully crafting poisoned outliers might have a serious impact.

Aside poisoning training sets, poisoning the database is another possibility, quite specific to content-based multimedia systems. We already shortly discussed this above, when presenting the possible goals of attackers in the security analysis. We demonstrated with our picture-in-picture attack that the success rate of false positives increases when the visual patches inserted in the attacked images come from the database probed by the similarity searches (see [Do et al., 2012a]). To the best of our knowledge, no one as yet started investigating this issue.

Research Direction #5: The Role of Encryption, Secrecy

Cryptography provides confidentiality, but none of its primitives comply with the variety of mathematical operations needed to describe, process, index and retrieve multimedia contents, while preserving efficiency at large scale. It is rather unclear whether the high level of security ensured by current secure systems (outside the field of CBIR systems) are indeed necessary when dealing with multimedia contents. It is key to understand whether it is safe or not to relax some assumptions. We in particular discussed the two-way versus one-way settings above, and what must be enforced to make one-way systems secure remains unclear.

The cryptographic protocols that are used today might be too strict in some cases when considering multimedia content-based retrieval applications. We have demonstrated, in a restricted setting, that increasing the level of ambiguity in the information is an alternative to encryption and secrecy: a contents server can hardly determine what a client indeed searches because the queries are "blurred" (see [Weng et al., 2014]) or because indexing is "blurred" (see [Furon et al., 2013, Mathon et al., 2013]). Additional studies along these lines of research are needed.

Research Direction #6: Secure Multimedia Description

Security is almost completely absent from the scientific horizon of the researchers designing description schemes for multimedia data. Their work focuses on compactness, richness, advanced sparse encoding techniques, robustness, invariance, extraction speed or recognition power at scale.

Enforcing security might mean to determine to what extend some kind of secrecy has to be included at description time, hoping it will not alter the otherwise very good properties of descriptors. To address this challenge, we think we need to sort what relates to robustness, security, secrecy and obscurity before being able to design new description schemes that are secure while good at recognizing contents.

Research Direction #7: Secure High-Dimensional Indexing and Retrieval

Researchers working on high-dimensional indexing and retrieval schemes focus on ways to better approximate neighborhood of points in high-dimensional spaces and to efficiently and effectively index and then search ever larger collections. These techniques have rarely been engaged from a security point of view where attackers try to severely distort this notion of neighborhood, making the search engine unable to find similar contents. Yet, we proved in preliminary studies that knowing the details of a real retrieval algorithm allows to break systems.

Everything is to invent here: methods to attack, to measure the severity of the problems, to counter-attack, ... Again, inventing goes along with the mandatory need to have solutions that remain efficient at very large scale. Designing secure, good and fast high-dimensional search techniques remains to be done.

Research Direction #8: Assembling & Evaluating Systems

Watermarking, cryptography, etc. are all tools dedicated to enforcing security. While each technology has its own merits, putting them all together does not necessarily create a fully secure system matching real-world application needs. Nobody knows today what are the pros and cons of each such tool when used with multimedia data, in particular security contexts, at very large scale, with demanding response times/throughput requirements. Our research aims at better understanding when these tools are somehow complementary, competitive or antagonistic. Building and then assessing the security of a complete system is thus very challenging.

VI.4. Other Perspectives Concluding Chapter VI

We discuss in this section some other perspectives that have not yet received as many attention as the ones described previously. The following discussions are short. This is absolutely not in relation with the energy we want to invest in addressing the problems they raise. It might even be the other way around, it is short only because time flies and so far not enough time could be dedicated to pushing these studies.

Working with multimedia sequences, that is, going beyond indexing still images is absolutely essential. Also, better understanding the curse of dimensionality problems thanks to the very recent intrinsic dimensionality indicators is central. We briefly go through these two topics before concluding.

VI.4.1. Multimedia Sequences

The main focus of this manuscript was on indexing and searching databases of still images. It is natural to wonder about working with multimedia sequences such as videos or audio documents. Such documents are everywhere, they are produced everyday and made available on-line by professional bodies such as TV channels, musical record labels and/or movie companies; they are also produced by people and accumulated by millions inside the repositories gathering user generated contents.

Compared to still images, the crucial differentiation is the *time* dimension of these audio and video documents. The applications involving such sequential documents are also richer. Video and audio streams, for example, have been under studies with the goal of detecting repeating ads, jingles, songs, etc, for structuring purposes, exploiting the visual modality (such as [Pua et al., 2004, Ide et al., 2004, Wu and Satoh, 2011]) or exploiting the audio modality (such as [Herley, 2006, Chen et al., 2011]).

Many approaches dealing with sequences heavily rely on traditional high-dimensional indexing techniques, such as the ones discussed in this manuscript. These approaches typically compute signatures from the frames of a video/audio stream before indexing the signatures. Searching for similar sequences then boils down to finding the frames that are similar between the query sequence and the ones from the database, combined with ad hoc and somewhat costly heuristics to ensure temporal consistency.

Dynamic Time Warping (DTW) is one algorithm of choice for dealing with the temporal dimension of sequences [Vintsyuk, 1968]. The DTW approach has been successfully applied to domains as diverse as economics, life sciences and bioinformatics, pattern recognition, monitoring, speech recognition, etc. DTW finds the optimal alignment between two given series of data. One of its strength is to cope with local distortions along the time dimension. DTW, however, is slow and costly to calculate as its time complexity is quadratic. In reaction, many contributions define approaches to accelerate DTW-based searches. The most popular approaches for speeding up DTW searches is to rely on lower bounding functions, see for example [Itakura, 1975, Sakoe and Chiba, 1978, Keogh and Ratanamahatana, 2005, Yi et al., 1998].

Simple functions lower bounding DTW distances accelerate searches by quickly pruning sequences that could not possibly be best matches. The tighter the bounds, the more they prune and the better the performance. Designing new functions that are even tighter is difficult because their computation is likely to become complex, canceling the benefits of their pruning. It is possible, however, to design simple functions with a higher pruning power by relaxing the *no false dismissal* assumption, resulting in approximate lower bound functions.

Romain Tavenard investigated this issue in details and designed such functions. His work

describes how the accuracy of approximations can be tuned, ranging from no false dismissal to potential losses when aggressively set for great response time savings. The interested reader can refer to his publications, [Tavenard et al., 2007a, Tavenard et al., 2007b, Tavenard et al., 2009, Tavenard, 2011, Tavenard and Amsaleg, 2013].

At very large scale, indexing is mandatory. Only very recently, indexing sequences after having turned them into symbolic representations has been considered (see for example [Burred, 2012, de Oliveira et al., 2013]), borrowing from seminal work in genomics and proteinomics.

In essence, searching withing multimedia sequences is very similar to motif search/discovery in DNA and genome sequences, a problem that has received tremendous attention. However, very different approaches have been considered in the two domains, mostly because of the very different nature of the data.

In bioinformatics, efficient algorithms have been proposed for motif search/discovery such as Blast [Altschul et al., 1990] or Plast [Nguyen and Lavenier, 2009]. Both rely on specific properties of DNA data, taking advantage of their symbolic nature to index short sequences of symbols (3 to 4 consecutive symbols), called n-grams. Distortions between occurrences of a motif follow rules specific to genomic (e.g., gaps in motifs), permitting efficient implementations.

It is therefore very tempting to explore how efficient motif search/discovery techniques from the field of genomic can be transposed to the multimedia field, taking into account the peculiarities of image and sound data. Research questions to address are for instance (but not limited to): How to quantize multimedia data in a meaningful manner? How to handle a large alphabet of symbols in Blast-like approaches? Are short n-grams meaningful for multimedia? How to exploit the semantics of symbols in multimedia? How to account for multimedia-specific temporal distortions?

VI.4.2. Intrinsic Dimensionality

High-dimensional indexing typically suffers from a phenomenon commonly referred to as the *curse of dimensionality*. As the dimensionality of the datasets increases, the discriminative power of similarity measures vanishes quickly, up to the point where methods that depend on them loose their effectiveness. When the dimensionality of a dataset is high, then similarity values tend to concentrate around their means [Weber et al., 1998, Beyer et al., 1999, Pestov, 2000].

Many techniques have been invented in an attempt to improve the discriminability of measures at high-dimensionality scales. These techniques have only a mitigated impact as they sometimes improve the behavior of this or that method, sometimes nothing changes. The situation is even more complicated because the same task performed on two different datasets having identical cardinalities and identical number of features may perform very differently.

Over the past decade or so, many indicators of the dimensionality of datasets have been proposed. Some rely on PCA and its variants, on multidimensional scaling, on fractal methods, on contrast-based indicators, etc. Overall, the target of these methods is to estimate distributions, possibly using parametric models, to discover the existence of local subspace dimensions or, more recently, to propose intrinsic dimensionality models such as the expansion dimension [Karger and Ruhl, 2002] and the generalized expansion dimension [Houle et al., 2012a]. These approaches determine the dimensionality at a particular point in space as a function of the rate at which the number of encountered data objects grows as the considered range of distances expands. These models start to be used in index structures, see [Beygelzimer et al., 2006, Houle et al., 2012b].

Intrinsic dimensionality estimators can be viewed as a measure capturing the complexity of a dataset. It is likely to help researchers and practitioners to get more insight into the nature of

their data, and therefore help them improve the efficiency and efficacy of their applications.

One possible use of intrinsic dimensionality estimators could be to automatically filter out some specific data items as noise, whenever they are associated with an unusually high intrinsic dimensionality value. In this way, the quality of query results may be enhanced as well.

The performance of content-based retrieval systems is usually assessed in terms of the precision and recall of queries on a ground truth data set. However, in high-dimensional settings it is often the case that some points are much less likely to appear in a query result than others. Intrinsic dimensionality estimators might be able to account for this difficulty. In turn, they might aid in the design of fair benchmarks that truly reflect the power of retrieval systems, according to a sound, mathematically-grounded procedure.

What indicators shall we use? How could they be computed? Current indicators are defined given a particular point in space. Can they be generalized to regions? How could operational rules for pruning, for better clustering, for performing outlier detection, etc. could be derived in practice from these indicators?

Chapter VII

General Conclusion

This manuscript has been giving an overview of several years of research aiming at inventing and evaluating systems and techniques for building efficient content-based retrieval systems managing extremely large databases of multimedia material. It has focused on the management of still images.

Most of the work described in this manuscript has been achieved while keeping in mind a database perspective. The most important notion that sheds a particular light on the high-dimensional indexing techniques we have been developing is the *mandatory need for secondary storage*. In essence, secondary storage is needed to cope with failures, which is a requirement for systems running in the real world. Secondary storage is also needed to handle truly large scale datasets.

This manuscript described several contributions. Some are easily identifiable, such as the NV-Tree and the DeCP distributed indexing scheme. Other contributions are more diffuse as they helped gaining some understanding of the problems we face.

Last, this manuscript presented several perspectives that are going beyond the strict limits of high-dimensional indexing.

There is still so much to do! So many interesting problems to look at! Perspectives are endless.

Bibliography

[Abaza et al., 2013] Abaza, A., Ross, A., Hebert, C., Harrison, M. A. F., and Nixon, M. S. (2013). A survey on ear biometrics. *ACM Computing Surveys*, 45(2). ⟨ p. 128 ⟩

[Adler, 2003] Adler, A. (2003). Sample images can be independently restored from face recognition templates. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, volume 2. ⟨ p. 128 ⟩

[Adler, 2004] Adler, A. (2004). Images can be regenerated from quantized biometric match score data. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, volume 1. ⟨ p. 128 ⟩

[Aghasaryan et al., 2013] Aghasaryan, A., Bouzid, M., Kostadinov, D., Kothari, M., and Nandi, A. (2013). On the use of LSH for privacy preserving personalization. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. ⟨ pp. 124, 136 ⟩

[Agrell et al., 2002] Agrell, E., Eriksson, T., Vardy, A., and Zeger, K. (2002). Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8). ⟨ p. 53 ⟩

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3). ⟨ p. 140 ⟩

[Amerini et al., 2013a] Amerini, I., Barni, M., Caldelli, R., and Costanzo, A. (2013). Counter-forensics of SIFT-based copy-move detection by means of keypoint classification. *EURASIP Journal on Image and Video Processing*, 2013. ⟨ p. 126 ⟩

[Amerini et al., 2013b] Amerini, I., Barni, M., Caldelli, R., and Costanzo, A. (2013). Removal and injection of keypoints for SIFT-based copy-move counter-forensics. *EURASIP Journal on Information Security*, 2013. ⟨ p. 126 ⟩

[Andoni and Indyk, 2006] Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*. ⟨ p. 60 ⟩

[Arigon et al., 2007] Arigon, A.-M., Miquel, M., and Tchounikine, A. (2007). Multimedia data warehouses: A multiversion model and a medical application. *Multimedia Tools and Applications*, 35(1). ⟨ p. 117 ⟩

[Barni, 2006] Barni, M. (2006). Processing encrypted signals: a new frontier for multimedia security. In *Proceedings of ACM International Workshop on Multimedia & Security*. ⟨ p. 129 ⟩

[Barni et al., 2010] Barni, M., Bianchi, T., Catalano, D., Di Raimondo, M., Labati, R., Failla, P., Fiore, D., Lazzeretti, R., Piuri, V., Piva, A., and Scotti, F. (2010). A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In *Proceedings of the IEEE International Conference on Biometrics: Theory Applications and Systems*. ⟨ p. 129 ⟩

[Barni et al., 2013] Barni, M., Kalker, T., and Katzenbeisser, S. (2013). Inspiring new research in the field of signal processing in the encrypted domain. *IEEE Signal Processing Magazine*, 30(2). ⟨ p. 129 ⟩

[Barreno et al., 2006] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure? In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. ⟨ p. 131 ⟩

[Barreno et al., 2010] Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81(2). ⟨ p. 131 ⟩

[Bartolini and Ciaccia, 2009] Bartolini, I. and Ciaccia, P. (2009). Integrating semantic and visual facets for browsing digital photo collections. In *Proceedings of the Italian Symposium on Advanced Database Systems*. ⟨ p. 117 ⟩

[Beekhof et al., 2012] Beekhof, F., Voloshynovskiy, S., and Farhadzadeh, F. (2012). Content authentication and identification under informed attacks. In *Proceedings of the IEEE International Workshop on Information Forensics and Security*. ⟨ p. 128 ⟩

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9). ⟨ p. 6 ⟩

[Berrani, 2004] Berrani, S.-A. (2004). *Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu*. PhD thesis, Université de Rennes 1. ⟨ p. 6 ⟩

[Berrani et al., 2002] Berrani, S.-A., Amsaleg, L., and Gros, P. (2002). Recherche par similarité dans les bases de données multidimensionnelles : panorama des techniques d'indexation. *RSTI–Ingénierie des systèmes d'information. Bases de données et multimédia*, 7(5). ⟨ p. 6 ⟩

[Beyer et al., 1999] Beyer, K. S., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is 'nearest neighbor' meaningful? In *Proceedings of the International Conference on Database Theory*. ⟨ pp. 6, 17, 140 ⟩

[Beygelzimer et al., 2006] Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbors. In *Proceedings of the International Conference on Machine Learning*. ⟨ p. 140 ⟩

[Bhatt and Kankanhalli, 2011] Bhatt, C. A. and Kankanhalli, M. S. (2011). Multimedia data mining: State of the art and challenges. *Multimedia Tools and Applications*, 51(1). ⟨ p. 122 ⟩

[Biggio et al., 2012] Biggio, B., Nelson, B., and Laskov, P. (2012). Poisoning attacks against support vector machines. In *Proceedings of the International Conference on Machine Learning*. ⟨ pp. 131, 134 ⟩

[Biggio et al., 2013] Biggio, B., Pillai, I., Rota Bulò, S., Ariu, D., Pelillo, M., and Roli, F. (2013). Is data clustering in adversarial settings secure? In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*. ⟨ pp. 131, 137 ⟩

[Böhm et al., 2001] Böhm, C., Berchtold, S., and Keim, D. A. (2001). Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3). ⟨ p. 6 ⟩

[Bohme and Kirchner, 2013] Bohme, R. and Kirchner, M. (2013). Counter-forensics: Attacking image forensics. In *Digital Image Forensics*. Springer. ⟨ p. 126 ⟩

[Boldi and Vigna, 2004] Boldi, P. and Vigna, S. (2004). The webgraph framework I: Compression techniques. In *Proceedings of the ACM International Conference on World Wide Web*. ⟨ p. 114 ⟩

[Bolze et al., 2006] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Touche, I. (2006). Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4). ⟨ p. 95 ⟩

[Bouganim et al., 2009] Bouganim, L., Jónsson, B. Þ., and Bonnet, P. (2009). uFLIP: Understanding flash io patterns. In *Proceedings of the Fourth Biennial Conference on Innovative Data Systems Research*. ⟨ p. 75 ⟩

[Bowyer et al., 2013] Bowyer, K. W., Hollingsworth, K., and Flynn, P. J. (2013). A survey of iris biometrics research: 2008-2010. In *Handbook of Iris Recognition*. Springer. ⟨ p. 128 ⟩

[Bringer et al., 2013] Bringer, J., Chabanne, H., and Patey, A. (2013). Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends. *IEEE Signal Processing Magazine*, 30(2). ⟨ p. 128 ⟩

[Bu et al., 2010] Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. (2010). Haloop: Efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1). ⟨ p. 104 ⟩

[Buhler, 2001] Buhler, J. (2001). Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5). ⟨ p. 32 ⟩

[Burred, 2012] Burred, J. (2012). Genetic motif discovery applied to audio analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 140 ⟩

[Caldelli et al., 2012] Caldelli, R., Amerini, I., Ballan, L., Serra, G., Barni, M., and Costanzo, A. (2012). On the effectiveness of local warping against SIFT-based copy-move detection. In *Proceedings of the IEEE Symposium on Communications, Control and Signal Processing*. ⟨ p. 126 ⟩

[Chang, 2011] Chang, E. Y. (2011). *Foundations of Large-Scale Multimedia Information Management and Retrieval: Mathematics of Perception*. Springer. ⟨ p. 131 ⟩

[Chen et al., 2011] Chen, J., Zhu, L., Feng, B., Ding, P., and Xu, B. (2011). A robust approach to mining repeated sequence in audio stream. In *Proceedings of the Conference of the International Speech Communication Association*. ⟨ pp. 123, 139 ⟩

[Cheng et al., 2006] Cheng, W., Pang, H., and Tan, K.-L. (2006). Authenticating multidimensional query results in data publishing. In *Data and Applications Security XX*, volume 4127 of *Lecture Notes in Computer Science*. Springer. ⟨ p. 136 ⟩

[Chierichetti et al., 2007] Chierichetti, F., Panconesi, A., Raghavan, P., Sozio, M., Tiberi, A., and Upfal, E. (2007). Finding near neighbors through cluster pruning. In *Proceedings of the ACM International Symposium on Principles of Database Systems*. ⟨ pp. 63, 64, 70, 79 ⟩

[Chor et al., 1995] Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. (1995). Private information retrieval. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*. ⟨ p. 132 ⟩

[Christlein et al., 2012] Christlein, V., Riess, C., Jordan, J., Riess, C., and Angelopoulou, E. (2012). An evaluation of popular copy-move forgery detection approaches. *IEEE Transactions on Information Forensics and Security*, 7(6). ⟨ p. 126 ⟩

[Chum and Matas, 2008] Chum, O. and Matas, J. (2008). Optimal randomized ransac. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8). ⟨ p. 16 ⟩

[Connelly and McNeill, 1999] Connelly, R. and McNeill, R. (1999). *The Multidimensional Manager*. Cognos. ⟨ p. 116 ⟩

[Conway and Sloane, 1982a] Conway, J. H. and Sloane, N. J. A. (1982). Fast quantizing and decoding and algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2). ⟨ p. 53 ⟩

[Conway and Sloane, 1982b] Conway, J. H. and Sloane, N. J. A. (1982). Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions on Information Theory*, 28(2). ⟨ p. 53 ⟩

[Conway and Sloane, 1987] Conway, J. H. and Sloane, N. J. A. (1987). *Sphere-packings, lattices, and groups*. Springer. ⟨ p. 53 ⟩

[Datar et al., 2004] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the ACM Symposium on Computational Geometry*. ⟨ pp. 29, 30, 65 ⟩

[Datta et al., 2008] Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2). ⟨ pp. 4, 6 ⟩

[de Oliveira et al., 2013] de Oliveira, L. S., do Patrocínio, Z. K. G., Guimarães, S. J. F., and Gravier, G. (2013). Searching for near-duplicate video sequences from a scalable sequence aligner. In *Proceedings of the IEEE International Symposium on Multimedia*. ⟨ p. 140 ⟩

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1). ⟨ pp. 92, 93 ⟩

[Delvinioti et al., 2014] Delvinioti, A., Jégou, H., Amsaleg, L., and Houle, E., M. (2014). Image retrieval with reciprocal and shared nearest neighbors. In *International Conference on Computer Vision Theory and Applications*. ⟨ pp. 109, 110 ⟩

[Diao et al., 2010] Diao, M., Mukherjea, S., Rajput, N., and Srivastava, K. (2010). Faceted search and browsing of audio content on spoken web. In *Proceedings of the ACM International Conference on Information and Knowledge Management*. ⟨ p. 117 ⟩

[Do, 2012] Do, T.-T. (2012). *Security analysis of image copy detection systems based on SIFT descriptors*. PhD thesis, Université de Rennes 1. ⟨ p. 130 ⟩

[Do et al., 2010a] Do, T.-T., Kijak, E., Furon, T., and Amsaleg, L. (2010). Challenging the security of content-based image retrieval systems. In *IEEE International Workshop on Multimedia Signal Processing*. ⟨ p. 130 ⟩

[Do et al., 2010b] Do, T.-T., Kijak, E., Furon, T., and Amsaleg, L. (2010). Deluding image recognition in SIFT-based CBIR systems. In *Proceedings of the ACM International Conference on Multimedia*, Workshop on Multimedia in Forensics, Security and Intelligence. ⟨ p. 130 ⟩

[Do et al., 2010c] Do, T.-T., Kijak, E., Furon, T., and Amsaleg, L. (2010). Understanding the security and robustness of SIFT. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 130 ⟩

[Do et al., 2012a] Do, T.-T., Amsaleg, L., Kijak, E., and Furon, T. (2012). Security-oriented picture-in-picture visual modifications. In *Proceedings of the ACM International Conference on Multimedia Retrieval*. ⟨ pp. 130, 133, 134, 137 ⟩

[Do et al., 2012b] Do, T.-T., Kijak, E., Amsaleg, L., and Furon, T. (2012). Enlarging hacker's toolbox: deluding image recognition by attacking keypoint orientations. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 130 ⟩

[Dong et al., 2011] Dong, W., Charikar, M., and Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the ACM International Conference on World Wide Web*. ⟨ p. 113 ⟩

[Douze et al., 2009] Douze, M., Jégou, H., Sandhawalia, H., Amsaleg, L., and Schmid, C. (2009). Evaluation of GIST descriptors for web-scale image search. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. ⟨ pp. 23, 65 ⟩

[Ekanayake et al., 2010] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., and Fox, G. (2010). Twister: a runtime for iterative MapReduce. In *Proceedings of the ACM International Symposium on High Performance Distributed Computing*. ⟨ p. 104 ⟩

[Engle et al., 2012] Engle, C., Lupher, A., Xin, R., Zaharia, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the ACM International Conference on Management of Data*. ⟨ p. 9 ⟩

[Ergün et al., 2000] Ergün, F., Kannan, S., Kumar, S., Rubinfeld, R., and Viswanathan, M. (2000). Spot-checkers. *Journal of Computer and System Sciences*, 60(3). ⟨ p. 136 ⟩

[Erkin et al., 2007] Erkin, Z., Piva, A., Katzenbeisser, S., Lagendijk, R., Shokrollahi, J., Neven, G., and Barni, M. (2007). Protection and retrieval of encrypted multimedia content: when cryptography meets signal processing. *EURASIP Journal on Information Security*, 2007. ⟨ p. 129 ⟩

[Erkin et al., 2009] Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., and Toft, T. (2009). Privacy-preserving face recognition. In *Proceedings of the International Symposium on Privacy Enhancing Technologies*. ⟨ pp. 128, 129 ⟩

[Eskevich et al., 2013] Eskevich, M., Jones, G. J., Chen, S., Aly, R., Ordelman, R., Nadeem, D., Guinaudeau, C., Gravier, G., Sébillot, P., Nies, T. D., Debevere, P., de Walle, R. V., Galuscakova, P., Pecina, P., and Larson, M. (2013). Multimedia information seeking through search and hyperlinking. In *Proceedings of the ACM International Conference on Multimedia Retrieval*. ⟨ p. 122 ⟩

[Fagin et al., 2003] Fagin, R., Kumar, R., and Sivakumar, D. (2003). Efficient similarity search and classification via rank aggregation. In *Proceedings of the ACM International Conference on Management of Data*. ⟨ pp. 34, 35 ⟩

[Fanti et al., 2013] Fanti, G., Finiasz, M., and Ramchandran, K. (2013). One-way private media search on public databases: The role of signal processing. *IEEE Signal Processing Magazine*, 30(2). ⟨ p. 132 ⟩

[Feigenbaum et al., 2002] Feigenbaum, J., Kannan, S., Strauss, M., and Viswanathan, M. (2002). Testing and spot-checking of data streams. *Algorithmica*, 34(1). ⟨ p. 136 ⟩

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6). ⟨ pp. 16, 109 ⟩

[Fraundorfer et al., 2007] Fraundorfer, F., Stewénius, H., and Nistér, D. (2007). A binning scheme for fast hard drive based image search. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 63, 64 ⟩

[Furon et al., 2013] Furon, T., Jégou, H., Amsaleg, L., and Mathon, B. (2013). Fast and secure similarity search in high dimensional space. In *Proceedings of the IEEE International Workshop on Information Forensics and Security*. ⟨ p. 137 ⟩

[Gal and Toledo, 2005] Gal, E. and Toledo, S. (2005). Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2). ⟨ p. 64 ⟩

[Ghemawat et al., 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ⟨ p. 93 ⟩

[Gionis et al., 1999] Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*. ⟨ pp. 27, 29, 32, 60 ⟩

[Gloe et al., 2007] Gloe, T., Kirchner, M., Winkler, A., and Böhme, R. (2007). Can we trust digital image forensics? In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 126 ⟩

[Gray et al., 1981] Gray, J., McJones, P., Blasgen, M., Lindsay, B., Lorie, R., Price, T., Putzolu, F., and Traiger, I. (1981). The recovery manager of the System R database manager. *ACM Computing Surveys*, 13(2). ⟨ p. 42 ⟩

[Gray and Neuhoff, 1998] Gray, R. M. and Neuhoff, D. L. (1998). Quantization. *IEEE Transactions on Information Theory*, 44(6). ⟨ pp. 28, 53, 62 ⟩

[Guha et al., 2011] Guha, S., Cheng, B., and Francis, P. (2011). Privad: Practical privacy in online advertising. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. ⟨ p. 136 ⟩

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM International Conference on Management of Data*. ⟨ p. 6 ⟩

[Guðmundsson, 2013] Guðmundsson, G. Þ. (2013). *Parallelism and Distribution for Very Large Scale Content Based Image Retrieval*. PhD thesis, Université de Rennes 1. ⟨ pp. 78, 105 ⟩

[Guðmundsson et al., 2010] Guðmundsson, G. Þ., Jónsson, B. Þ., and Amsaleg, L. (2010). A large-scale performance study of cluster-based high-dimensional indexing. In *Proceedings of the ACM International Conference on Multimedia*, Workshop on Very-Large-Scale Multimedia Corpus, Mining and Retrieval. ⟨ p. 72 ⟩

[Guðmundsson et al., 2012] Guðmundsson, G. Þ., Amsaleg, L., and Jónsson, B. Þ. (2012). Impact of storage technology on the efficiency of cluster-based high-dimensional index creation. In *Proceedings of the International Conference on Database Systems for Advanced Applications*. ⟨ p. 75 ⟩

[Hamzaoui et al., 2014] Hamzaoui, A., Letessier, P., Joly, A., Buisson, O., and Boujemaa, N. (2014). Object-based visual query suggestion. *Multimedia Tools and Applications*, 68(2). ⟨ p. 110 ⟩

[Harðarson and Jónsson, 2007] Harðarson, K. and Jónsson, B. Þ. (2007). Breaking out of the shoebox: Towards having fun with digital images. In *Proceedings of the International Workshop on Computer Vision Meets Databases*. ⟨ p. 117 ⟩

[Herley, 2006] Herley, C. (2006). Argos: automatically extracting repeating objects from multimedia streams. *IEEE Transactions on Multimedia*, 8(1). ⟨ p. 139 ⟩

[Houle, 2008] Houle, M. E. (2008). The relevant-set correlation model for data clustering. *Statistical Analysis and Data Mining*, 1(3). ⟨ pp. 110, 111 ⟩

[Houle et al., 2010] Houle, M. E., Kriegel, H. P., Kröger, P., Schubert, E., and Zimek, A. (2010). Can shared-neighbor distances defeat the curse of dimensionality? In *Proceedings of the International Conference on Scientific and Statistical Database Management*. ⟨ pp. 110, 111 ⟩

[Houle et al., 2012a] Houle, M. E., Kashima, H., and Nett, M. (2012). Generalized expansion dimension. In *Proceedings of the IEEE International Conference on Data Mining Workshops*. ⟨ p. 140 ⟩

[Houle et al., 2012b] Houle, M. E., Ma, X., Nett, M., and Oria, V. (2012). Dimensional testing for multi-step similarity search. In *Proceedings of the International Conference on Machine Learning*. ⟨ p. 140 ⟩

[Hsu et al., 2009] Hsu, C.-Y., Lu, C.-S., and Pei, S.-C. (2009). Secure and robust SIFT. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ pp. 128, 130 ⟩

[Hsu et al., 2010] Hsu, C.-Y., Lu, C.-S., and Pei, S.-C. (2010). Secure and robust SIFT with resistance to chosen-plaintext attack. In *Proceedings of the IEEE International Conference on Image Processing*. ⟨ p. 128 ⟩

[Huang et al., 2011a] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. D. (2011). Adversarial machine learning. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*. ⟨ p. 131 ⟩

[Huang et al., 2011b] Huang, Y., Malka, L., Evans, D., and Katz, J. (2011). Efficient privacy-preserving biometric identification. In *Proceedings of the Network and Distributed System Security Symposium*. ⟨ p. 129 ⟩

[Ide et al., 2004] Ide, I., Mo, H., Katayama, N., and Satoh, S. (2004). Topic threading for structuring a large-scale news video archive. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. ⟨ p. 139 ⟩

[Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing*. ⟨ pp. 7, 30 ⟩

[Isard et al., 2007] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Operating System Review*, 41(3). ⟨ pp. 9, 92 ⟩

[Itakura, 1975] Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 139 ⟩

[Jain et al., 2004] Jain, A., Ross, A., and Prabhakar, S. (2004). An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1). ⟨ p. 128 ⟩

[Jain and Nandakumar, 2012] Jain, A. K. and Nandakumar, K. (2012). Biometric authentication: System security and user privacy. *IEEE Computer*, 45(11). ⟨ p. 128 ⟩

[Jégou et al., 2007] Jégou, H., Harzallah, H., and Schmid, C. (2007). A contextual dissimilarity measure for accurate and efficient image search. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ p. 109 ⟩

[Jégou et al., 2008a] Jégou, H., Douze, M., and Schmid, C. (2008). Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of the European Conference on Computer Vision*. ⟨ pp. 109, 113 ⟩

[Jégou et al., 2008b] Jégou, H., Amsaleg, L., Schmid, C., and Gros, P. (2008). Query-adaptative locality sensitive hashing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 60 ⟩

[Jégou et al., 2010a] Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 8, 65, 68, 109 ⟩

[Jégou et al., 2010b] Jégou, H., Douze, M., and Schmid, C. (2010). Improving bag-of-features for large scale image search. *International Journal on Computer Vision*, 87(3). ⟨ pp. 64, 65, 68 ⟩

[Jégou et al., 2011] Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1). ⟨ pp. 7, 62, 79 ⟩

[Jégou et al., 2012] Jégou, H., Perronnin, F., Douze, M., Sanchez, J., Pérez, P., and Schmid, C. (2012). Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9). ⟨ pp. 35, 79 ⟩

[Jin et al., 2010] Jin, X., Han, J., Cao, L., Luo, J., Ding, B., and Lin, C. X. (2010). Visual cube and on-line analytical processing of images. In *Proceedings of the ACM International Conference on Information and Knowledge Management*. ⟨ p. 117 ⟩

[Johnson and Ramchandran, 2003] Johnson, M. and Ramchandran, K. (2003). Dither-based secure image hashing using distributed coding. In *Proceedings of the IEEE International Conference on Image Processing*. ⟨ p. 135 ⟩

[Joly and Buisson, 2008] Joly, A. and Buisson, O. (2008). A posteriori multi-probe locality sensitive hashing. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ pp. 33, 65, 79 ⟩

[Jónsson et al., 2014] Jónsson, B. Þ., Eiríksdóttir, Á., Waage, Ó., Tómasson, G., Sigurþórsson, H., and Amsaleg, L. (2014). $M^3 + P^3 + O^3$ = multi-D photo browsing. In *Proceedings of the International Conference on MultiMedia Modeling*. ⟨ p. 117 ⟩

[Karger and Ruhl, 2002] Karger, D. R. and Ruhl, M. (2002). Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the ACM Symposium on Theory of Computing*. ⟨ p. 140 ⟩

[Kay, 2012] Kay, R. (2012). Exploring the use of video podcasts in education: A comprehensive review of the literature. *Computers in Human Behavior*, 28(3). ⟨ p. 122 ⟩

[Keogh and Ratanamahatana, 2005] Keogh, E. and Ratanamahatana, C. (2005). Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3). ⟨ p. 139 ⟩

[Kerckhoffs, 1883] Kerckhoffs, A. (1883). La cryptographie militaire. *Journal des sciences militaires*, 9. ⟨ p. 127 ⟩

[Kozat et al., 2004] Kozat, S., Venkatesan, R., and Mihcak, K. (2004). Robust perceptual image hashing via matrix invariants. In *Proceedings of the IEEE International Conference on Image Processing*. ⟨ p. 135 ⟩

[Lagendijk et al., 2013] Lagendijk, R., Erkin, Z., and Barni, M. (2013). Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. *IEEE Signal Processing Magazine*, 30(1). ⟨ pp. 129, 131 ⟩

[Laskov and Lippmann, 2010] Laskov, P. and Lippmann, R. (2010). Machine learning in adversarial environments. *Machine Learning*, 81(2). ⟨ p. 131 ⟩

[Law-To et al., 2007] Law-To, J., Chen, L., Joly, A., Laptev, I., Buisson, O., Gouet-Brunet, V., Boujemaa, N., and Stentiford, F. (2007). Video copy detection: a comparative study. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. ⟨ p. 124 ⟩

[Law-To et al., 2010] Law-To, J., Grefenstete, G., Gauvain, J.-L., Gravier, G., Lamel, L., and Despres, J. (2010). VoxaleadNews: Robust automatic segmentation of video content into browsable and searchable subjects. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 122 ⟩

[Lee et al., 2008] Lee, S.-W., Moon, B., Park, C., Kim, J.-M., and Kim, S.-W. (2008). A case for flash memory ssd in enterprise database applications. In *Proceedings of the ACM International Conference on Management of Data*. ⟨ p. 64 ⟩

[Lejsek et al., 2005] Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2005). Efficient and effective image copyright enforcement. In *Actes des Journées sur les bases de données avancées*. ⟨ pp. 35, 38, 49, 124 ⟩

[Lejsek et al., 2006a] Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2006). Scalability of local image descriptors: a comparative study. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ pp. 35, 48 ⟩

[Lejsek et al., 2006b] Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2006). Blazingly fast image copyright enforcement. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ pp. 38, 124 ⟩

[Lejsek et al., 2009] Lejsek, H., Ásmundsson, F. H., Jónsson, B. Þ., and Amsaleg, L. (2009). NV-Tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5). ⟨ pp. 33, 38, 39, 42, 48 ⟩

[Lejsek et al., 2011] Lejsek, H., Jónsson, B. Þ., and Amsaleg, L. (2011). NV-Tree: nearest neighbors at the billion scale. In *Proceedings of the ACM International Conference on Multimedia Retrieval*. ⟨ pp. 7, 38, 42, 44 ⟩

[Li et al., 2008] Li, F., Hadjieftheriou, M., Kollios, G., and Reyzin, L. (2008). Authenticated index structures for outsourced databases. In *Handbook of Database Security*. Springer. ⟨ p. 136 ⟩

[Liu and Wang, 2012] Liu, L. and Wang, L. (2012). What has my classifier learned? visualizing the classification rules of bag-of-feature model by support region detection. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 130, 131 ⟩

[Low et al., 2012] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. (2012). Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8). ⟨ pp. 9, 92 ⟩

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal on Computer Vision*, 60(2). ⟨ pp. 4, 7, 18, 68 ⟩

[Lu and Hsu, 2012] Lu, C.-S. and Hsu, C.-Y. (2012). Constraint-optimized keypoint inhibition/insertion attack: Security threat to scale-space image feature extraction. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 130 ⟩

[Lu et al., 2009] Lu, W., Varna, A., Swaminathan, A., and Wu, M. (2009). Secure image retrieval through feature protection. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 128 ⟩

[Lv et al., 2007] Lv, Q., Josephson, W., Wang, Z., Charikar, M., and Li, K. (2007). Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the International Conference on Very Large Data Bases*. ⟨ pp. 33, 65, 79 ⟩

[Machanavajjhala et al., 2007] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). L-diversity: Privacy beyond K-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1). ⟨ p. 136 ⟩

[Malkin and Venkatesan, 2004] Malkin, M. and Venkatesan, R. (2004). The randlet transform: application to universal perceptual hashing and image identification. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*. ⟨ p. 135 ⟩

[Martel et al., 2004] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., and Stubblebine, S. G. (2004). A general model for authenticated data structures. *Algorithmica*, 39(1). ⟨ p. 136 ⟩

[Mathon et al., 2013] Mathon, B., Furon, T., Amsaleg, L., and Bringer, J. (2013). Secure and efficient approximate nearest neighbors search. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*. ⟨ p. 137 ⟩

[Mejdoub et al., 2009] Mejdoub, M., Fonteles, L., BenAmar, C., and Antonini, M. (2009). Embedded lattices tree: An efficient indexing scheme for content based retrieval on image databases. *Journal of Visual Communication and Image Representation*, 20(2). Special issue on Emerging Techniques for Multimedia Content Sharing, Search and Understanding. ⟨ p. 55 ⟩

[Melloni et al., 2013] Melloni, A., Bestagini, P., Costanzo, A., Barni, M., Tagliasacchi, M., and Tubaro, S. (2013). Attacking image classification based on bag-of-visual-words. In *Proceedings of the IEEE International Workshop on Information Forensics and Security*. ⟨ p. 130 ⟩

[Merkt et al., 2011] Merkt, M., Weigand, S., Heier, A., and Schwan, S. (2011). Learning with videos vs. learning with print: The role of interactive features. *Learning & Instruction*, 21(6). ⟨ p. 122 ⟩

[Mikolajczyk and Schmid, 2004] Mikolajczyk, K. and Schmid, C. (2004). Scale and affine invariant interest point detectors. *International Journal on Computer Vision*, 60(1). ⟨ p. 68 ⟩

[Mohan et al., 1992] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. (1992). Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1). ⟨ pp. 10, 42 ⟩

[Moise et al., 2013] Moise, D., Shestakov, D., Guðmundsson, G. Þ., and Amsaleg, L. (2013). Indexing and searching 100M images with Map-Reduce. In *Proceedings of the ACM International Conference on Multimedia Retrieval*. ⟨ pp. 72, 78 ⟩

[Moon et al., 2001] Moon, B., Jagadish, H., Faloutsos, C., and Saltz, J. H. (2001). Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1). ⟨ p. 53 ⟩

[Muja and Lowe, 2009] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*. ⟨ p. 65 ⟩

[Nguyen and Lavenier, 2009] Nguyen, V. H. and Lavenier, D. (2009). Plast: parallel local alignment search tool for database comparison. *BMC Bioinformatics*, 10. ⟨ p. 140 ⟩

[Nistér and Stewénius, 2006] Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 61, 62, 63, 65, 109 ⟩

[Oliva and Torralba, 2001] Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal on Computer Vision*, 42(3). ⟨ pp. 35, 68 ⟩

[Osadchy et al., 2010] Osadchy, M., Pinkas, B., Jarrous, A., and Moskovich, B. (2010). Scifi - a system for secure face identification. In *Proceedings of the IEEE Symposium on Security and Privacy*. ⟨ p. 128 ⟩

[Over et al., 2011] Over, P., Awad, G., Michel, M., Fiscus, J., Kraaij, W., and Smeaton, A. F. (2011). An overview of the goals, tasks, data, evaluation mechanisms and metrics. In *TRECVID Workshop, NIST*. ⟨ p. 129 ⟩

[Owen et al., 2011] Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action*. Manning Publications Co. ⟨ p. 104 ⟩

[Panigrahy, 2006] Panigrahy, R. (2006). Entropy based nearest neighbor search in high dimensions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithm*. ⟨ p. 33 ⟩

[Park and Jain, 2010] Park, U. and Jain, A. K. (2010). Face matching and retrieval using soft biometrics. *IEEE Transactions on Information Forensics and Security*, 5(3). ⟨ p. 128 ⟩

[Paulevé, 2008] Paulevé, L. (2008). Réseaux géométriques pour l'indexation et la recherche en grandes dimensions. Master's thesis, Université de Rennes 1. ⟨ p. 57 ⟩

[Paulevé et al., 2010] Paulevé, L., Jégou, H., and Amsaleg, L. (2010). Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11). ⟨ pp. 56, 79 ⟩

[Perronnin et al., 2010] Perronnin, F., Liu, Y., Sánchez, J., and Poirier, H. (2010). Large-scale image retrieval with compressed fisher vectors. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ p. 109 ⟩

[Pestov, 2000] Pestov, V. (2000). On the geometry of similarity search: Dimensionality curse and concentration of measure. *Information Processing Letters*, 73(1–2). ⟨ p. 140 ⟩

[Petitcolas et al., 1998] Petitcolas, F. A. P., Anderson, R. J., and Kuhn, M. G. (1998). Attacks on copyright marking systems. In *Proceedings of the International Workshop on Information Hiding*. ⟨ pp. 18, 22 ⟩

[Petitcolas et al., 2001] Petitcolas, F. A. P., Steinebach, M., Raynal, F., Dittmann, J., Fontaine, C., and Fates, N. (2001). Public automated web-based evaluation service for watermarking schemes: Stirmark benchmark. In *Proceedings of the SPIE*, volume 4314. ⟨ pp. 18, 22 ⟩

[Petrushin and Khan, 2007] Petrushin, V. A. and Khan, L. (2007). *Multimedia Data Mining and Knowledge Discovery*. Springer. ⟨ p. 122 ⟩

[Philbin et al., 2007] Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 61, 62, 113 ⟩

[Philbin et al., 2008] Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2008). Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 57, 62, 71, 79, 113 ⟩

[Poulisse et al., 2014] Poulisse, G. J., Patsis, Y., and Moens, M.-F. (2014). Unsupervised scene detection and commentator building using multi-modal chains. *Multimedia Tools and Applications*, 70(1). ⟨ p. 122 ⟩

[Pua et al., 2004] Pua, K. M., Gauch, J. M., Gauch, S. E., and Miadowicz, J. Z. (2004). Real time repeated video sequence identification. *Computer Vision and Image Understanding*, 93(3). ⟨ p. 139 ⟩

[Qin et al., 2011] Qin, D., Gammeter, S., Bossard, L., Quack, T., and Gool, L. V. (2011). Hello neighbor: accurate object retrieval with k-reciprocal nearest neighbors. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ pp. 110, 112 ⟩

[Rane and Boufounos, 2013] Rane, S. and Boufounos, P. (2013). Privacy-preserving nearest neighbor methods: comparing signals without revealing them. *IEEE Signal Processing Magazine*, 30(2). ⟨ p. 129 ⟩

[Rane et al., 2013] Rane, S., Wang, Y., Draper, S. C., and Ishwar, P. (2013). Secure biometrics: Concepts, authentication architectures, and challenges. *IEEE Signal Processing Magazine*, 30(5). ⟨ p. 128 ⟩

[Rathgeb and Uhl, 2011] Rathgeb, C. and Uhl, A. (2011). A survey on biometric cryptosystems and cancelable biometrics. *EURASIP Journal on Information Security*, 2011. ⟨ p. 128 ⟩

[Robinson et al., 2013] Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media. ⟨ p. 114 ⟩

[Rohou, 2011] Rohou, E. (2011). Tiptop: Hardware Performance Counters for the Masses. Rapport de recherche RR-7789, INRIA. ⟨ p. 88 ⟩

[Rúnarsson, 2011] Rúnarsson, K. (2011). A face recognition plug-in for the PhotoCube browser. Master's thesis, Reykjavik University. ⟨ p. 118 ⟩

[Sadeghi et al., 2010] Sadeghi, A.-R., Schneider, T., and Wehrenberg, I. (2010). Efficient privacy-preserving face recognition. In *Proceedings of the International Conference on Information Security and Cryptology*. ⟨ pp. 128, 129 ⟩

[Sakoe and Chiba, 1978] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 139 ⟩

[Samet, 2007] Samet, H. (2007). *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann. ⟨ p. 6 ⟩

[Sebe et al., 2008] Sebe, N., Tian, Q., Lew, M. S., and Huang, T. S. (2008). Similarity matching in computer vision and multimedia. *Computer Vision and Image Understanding*, 110(3). ⟨ p. 4 ⟩

[Shaft and Ramakrishnan, 2006] Shaft, U. and Ramakrishnan, R. (2006). Theory of nearest neighbors indexability. *ACM Transactions on Database Systems*, 31(3). ⟨ p. 17 ⟩

[Shakhnarovich et al., 2006] Shakhnarovich, G., Darrell, T., and Indyk, P. (2006). *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, chapter 3. MIT Press. ⟨ pp. 29, 30, 54, 57 ⟩

[Shashank et al., 2008] Shashank, J., Kowshik, P., Srinathan, K., and Jawahar, C. (2008). Private content based image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ p. 132 ⟩

[Shestakov et al., 2013] Shestakov, D., Moise, D., Guðmundsson, G. Þ., and Amsaleg, L. (2013). Scalable high-dimensional indexing with Hadoop. In *International Workshop on Content-Based Multimedia Indexing*. ⟨ pp. 72, 78 ⟩

[Shvachko et al., 2010] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop distributed file system. In *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies*. ⟨ p. 94 ⟩

[Sigurðardottir et al., 2005] Sigurðardottir, R., Hauksson, H., Jónsson, B. Þ., and Amsaleg, L. (2005). A case study of the quality vs. time trade-off for approximate image descriptor search. In *Proceedings of the IEEE International Conference on Data Engineering*, Workshop on Managing Data for Emerging Multimedia Applications. ⟨ p. 71 ⟩

[Sigurþórsson, 2011] Sigurþórsson, H. (2011). PhotoCube: Multi-dimensional image browsing. Master's thesis, Reykjavik University. ⟨ pp. 118, 121 ⟩

[Sion, 2005] Sion, R. (2005). Query execution assurance for outsourced databases. In *Proceedings of the International Conference on Very Large Data Bases*. ⟨ p. 136 ⟩

[Sivic and Zisserman, 2003] Sivic, J. and Zisserman, A. (2003). Video google: A text retrieval approach to object matching in videos. In *Proceedings of the IEEE International Conference on Computer Vision*. ⟨ pp. 7, 58, 61, 65, 68, 79 ⟩

[Snoek and Worring, 2005] Snoek, C. and Worring, M. (2005). Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools and Applications*, 25(1). ⟨ p. 122 ⟩

[Swaminathan et al., 2006] Swaminathan, A., Mao, Y., and Wu, M. (2006). Robust and secure image hashing. *IEEE Transactions on Information Forensics and Security*, 1(2). ⟨ p. 135 ⟩

[Sweeney, 2002] Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5). ⟨ p. 136 ⟩

[Szeliski, 2010] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer. ⟨ pp. 4, 109 ⟩

[Ta and Gravier, 2012] Ta, A.-P. and Gravier, G. (2012). Unsupervised mining of multiple audiovisually consistent clusters for video structure analysis. In *Proceedings of the IEEE International Conference on Multimedia and Exhibition*. ⟨ p. 123 ⟩

[Tao and Papadias, 2004] Tao, Y. and Papadias, D. (2004). Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(12). ⟨ p. 136 ⟩

[Tavenard, 2011] Tavenard, R. (2011). *Indexation de séquences de descripteurs*. PhD thesis, Université de Rennes 1. ⟨ p. 140 ⟩

[Tavenard and Amsaleg, 2013] Tavenard, R. and Amsaleg, L. (2013). Improving the efficiency of traditional DTW accelerators. *Knowledge and Information Systems*. ⟨ p. 140 ⟩

[Tavenard et al., 2007a] Tavenard, R., Amsaleg, L., and Gravier, G. (2007). Estimation de similarité entre séquences de descripteurs à l'aide de machines à vecteurs supports. In *Actes des Journées sur les bases de données avancées*. ⟨ p. 140 ⟩

[Tavenard et al., 2007b] Tavenard, R., Amsaleg, L., and Gravier, G. (2007). Machines à vecteurs supports pour la comparaison de séquences de descripteurs. In *Actes des Journées compression et représentation des signaux audiovisuels*. ⟨ p. 140 ⟩

[Tavenard et al., 2009] Tavenard, R., Amsaleg, L., and Gravier, G. (2009). Model-based similarity estimation of multidimensional temporal sequences. *Annals of Telecommunications*, 64(5). ⟨ p. 140 ⟩

[Tavenard et al., 2011] Tavenard, R., Jégou, H., and Amsaleg, L. (2011). Balancing clusters to reduce response time variability in large scale image search. In *International Workshop on Content-Based Multimedia Indexing*. ⟨ p. 65 ⟩

[Tómasson, 2011] Tómasson, G. (2011). ObjectCube – A generic multi-dimensional model for media browsing. Master's thesis, Reykjavík University. ⟨ p. 117 ⟩

[Tómasson et al., 2011] Tómasson, G., Sigurþórsson, H., Jónsson, B. Þ., and Amsaleg, L. (2011). Photocube: Effective and efficient multi-dimensional browsing of personal photo collections. In *Proceedings of the ACM International Conference on Multimedia Retrieval*. ⟨ p. 118 ⟩

[Tómasson et al., 2012a] Tómasson, G., Jónsson, B. Þ., and Amsaleg, L. (2012). Objectcube: A novel data model for multi-dimensional media browsing. Technical Report RUTR-CS12002, Reykjavik University. ⟨ pp. 117, 118 ⟩

[Tómasson et al., 2012b] Tómasson, G., Sigurþórsson, H., Jónsson, B. Þ., and Amsaleg, L. (2012). Photocube: Towards multi-dimensional image browsing. Technical Report RUTR-CS12001, Reykjavik University. ⟨ p. 118 ⟩

[Toubiana et al., 2010] Toubiana, V., Narayanan, A., Boneh, D., Nissenbaum, H., and Barocas, S. (2010). Adnostic: Privacy preserving targeted advertising. In *Proceedings of the Network and Distributed System Security Symposium*. ⟨ p. 136 ⟩

[Van Mulbregt et al., 1998] Van Mulbregt, P., Carp, I., Gillick, L., Lowe, S., and Yamron, J. (1998). Text segmentation and topic tracking on broadcast news via a hidden markov model approach. In *Proceedings of the International Conference on Speech and Language Processing*. ⟨ p. 122 ⟩

[Venugopalan and Savvides, 2011] Venugopalan, S. and Savvides, M. (2011). How to generate spoofed irises from an iris code template. *IEEE Transactions on Information Forensics and Security*, 6(2). ⟨ p. 128 ⟩

[Vintsyuk, 1968] Vintsyuk, T. (1968). Speech discrimination by dynamic programming. *Cybernetics*, 4(1). ⟨ p. 139 ⟩

[Wang et al., 2012] Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., and Li, S. (2012). Scalable k-nn graph construction for visual descriptors. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ p. 114 ⟩

[Weber et al., 1998] Weber, R., Schek, H.-J., and Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Data Bases*. ⟨ p. 140 ⟩

[Weinzaepfel et al., 2011] Weinzaepfel, P., Jégou, H., and Pérez, P. (2011). Reconstructing an image from its local descriptors. In *Proceedings of the IEEE International Conference on Computer Vision & Pattern Recognition*. ⟨ p. 131 ⟩

[Weng et al., 2014] Weng, L., Amsaleg, L., Morton, A., and Marchand-Maillet, S. (2014). A privacy-preserving framework for large scale content-based information retrieval. Technical Report CVML-TR.1401, University of Geneva. ⟨ pp. 132, 133, 137 ⟩

[Worring and Koelma, 2013] Worring, M. and Koelma, D. C. (2013). Multimedia pivot tables. In *Proceedings of Visual Analytics Science and Technology*. ⟨ p. 117 ⟩

[Wu and Satoh, 2011] Wu, X. and Satoh, S. (2011). Temporal recurrence hashing algorithm for mining commercials from multimedia streams. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. ⟨ p. 139 ⟩

[Yee et al., 2003] Yee, K.-P., Swearingen, K., Li, K., and Hearst, M. (2003). Faceted metadata for image search and browsing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ⟨ p. 117 ⟩

[Yi et al., 1998] Yi, B., Jagadish, H. V., and Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. In *Proceedings of the IEEE International Conference on Data Engineering*. ⟨ p. 139 ⟩

[Yizhi et al., 2010] Yizhi, L., Shouxun, L., Sheng, T., and Yongdong, Z. (2010). Adult image detection combining bovw based on region of interest and color moments. In *Intelligent Information Processing V*, volume 340 of *IFIP Advances in Information and Communication Technology*. Springer. ⟨ p. 130 ⟩

[Yuan et al., 2012] Yuan, J., Gravier, G., Campion, S., Li, X., and Jégou, H. (2012). Efficient mining of repetitions in large-scale TV streams with product quantization hashing. In *Proceedings of the European Conference on Computer Vision*, Workshop on Web-scale Vision and Social Media. ⟨ p. 123 ⟩

[Zaharia et al., 2010] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: cluster computing with working sets. In *Proceedings of the USENIX Conference on Hot Topics in Cloud Computing*. ⟨ pp. 9, 104 ⟩

[Zaïane et al., 1998] Zaïane, O. R., Han, J., Li, Z.-N., and Hou, J. (1998). Mining multimedia data. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*. ⟨ p. 117 ⟩

[Zhang et al., 2010a] Zhang, T., Chao, H., Willis, C., and Tretter, D. (2010). Consumer image retrieval by estimating relation tree from family photo collections. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. ⟨ p. 135 ⟩

[Zhang et al., 2010b] Zhang, W., Gao, K., Zhang, Y.-D., and Li, J.-T. (2010). Data-oriented locality sensitive hashing. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 33 ⟩

[Zhao et al., 2013] Zhao, W., Jégou, H., and Gravier, G. (2013). Sim-min-hash: An efficient matching technique for linking large image collections. In *Proceedings of the ACM International Conference on Multimedia*. ⟨ p. 122 ⟩